UNIVERSITY OF PISA
DEPARTMENT OF INFORMATION ENGINEERING

Master Thesis in Electronic Engineering

Modelling of Li-Ion Batteries for Portable Electronics in Analog and Digital Simulation Environments

Candidate:

Michele Rognini

Supervisors:

Prof. Roberto Saletti
Prof. Federico Baronti

Advisor:

Andrea Barsanti

Academic Year 2015/2016

Contents

List of Figures

List of Tables

1 Introduction

Nowadays rechargeable lithium-ion batteries are considered the batteries of choice for many portable electronic devices. The reason of their success lies in their high energy and power density, no memory effect and low self-discharge rate. Consequently, lithium-ion batteries are continuously spreading thanks to the driving force represented by the growth of the portable devices market. The resulting cost reduction, combined with the superior characteristics that this technology can offer, suggest them as energy accumulation devices basically in all the portable electronic devices, from mobile phones to tablets and laptop computers.

Unfortunately, the potential offered by the lithium chemistry are counterbalanced by some critical factors [1]. Over-charging or over-discharging a Li-ion battery reduces drastically battery lifetime and may even cause permanent damages. Damages can also occur if the battery operates out of its temperature range. These dangerous conditions make the use of battery management systems and smart chargers necessary. The main objective of the battery management systems is to ensure the optimum and safest use of the energy in the battery and to extend its lifetime by preventing any risk of damage. This goal can be achieved by monitoring and controlling the charging and discharging processes of the battery in order to ensure that the battery works in its safe operating ranges.

Hence, the design of such systems represents a critical aspect in the project of the whole portable device. Malfunctions of the battery management system can cause serious damages to the battery with the annexes risks for the electronic device user. Consequently, the availability of an accurate model capable of representing the dynamic behaviour of a Li-ion battery to be used in system-level simulations is important for the design and the optimization of the battery management systems.

The goal of this thesis is to develop an equivalent model of a Li-ion battery to be used in multi-domain simulations, aimed at the design and at the optimization of battery management systems and battery chargers.

A complete modelling process is realized for a portable electronic device Li-ion battery. The battery of the Samsung Galaxy Note 4 smartphone is used as example.

The thesis has been carried out in collaboration with Dialog Semiconductor. The work is organized follow. Firstly, an introduction on the lithium-ion battery technology is provided and the main Li-ion modelling techniques are presented.

After that, the reference battery model is chosen. In order to be easily implemented in different simulation environments, the reference model is an equivalent circuit model.

Once the reference battery model has been chosen, the battery needs to be characterized with a well-defined experiment commonly used to extract information about the battery behaviour.

Then, the parameters of the reference battery equivalent model have to be extracted using an estimation algorithm from the characterization test.

The parameter estimation task is performed using firstly a conventional method and then an optimized parameter estimation technique is introduced to overcome the

limits of the conventional method. The parameter estimation algorithm represents doubtless one of the more innovative aspects of this thesis.

Once that the parameters have been extracted, three different versions of the battery model are implemented in different simulation environments. The first model is realized in a high-level environment such as Matlab/Simulink. The second model is implemented as an analog electrical circuit in the Cadence Virtuoso environment. The third model is a digital/mixed-signal model and it is implemented using SystemVerilog language; its simulations are performed in the Cadence Incisive environment.

After the battery model implementation, a validation is performed on the three battery models in order to verify the correctness of their behaviour by comparing the simulations to the measures.

A root mean square error less than 0.3 % and a 0.8 % maximum error on the voltage show that the implemented models are able to predict the battery behaviour accurately, in all the three different simulation environments.

So, the battery models are ready to be used for the design of battery management systems, battery chargers or, in general, battery-powered systems. The models can also be easily extended to other battery technologies.

2 Background

2.1 Li-ion battery overview

2.1.1 Basic characteristics

A battery basically converts chemical energy into electric energy. This process is irreversible in primary, or non-rechargeable batteries, while is reversible in secondary, or rechargeable batteries. Lithium batteries can be divided into three different categories: lithium metal, lithium-ion and lithium-ion polymer. Lithium metal batteries are primary batteries, while both lithium-ion and lithium-ion polymer batteries are rechargeable.

Let's focus attention on the lithium-ion technology. In the following, some basic definitions are provided in order to describe the general characteristics and the typical behaviour of lithium-ion batteries.

The terminal voltage is the voltage that exists between the battery external terminals. The terminal voltage varies with the operating conditions of the battery.

The capacity of a battery corresponds to the amount of charge that can be stored stored in the battery. The nominal capacity is represented by the amount of charge that a battery can deliver to a load from full charge in a well-defined condition, specified by the battery manufacturer. The available capacity of a battery depends upon the rate at which it is discharged. If a battery is discharged at a relatively high rate, the available capacity will be lower than the nominal capacity.

The end-of-discharge (EOD) and the end-of-charge (EOC) conditions, for a battery, represent respectively the voltages when discharge and charge are considered complete. They are also called cut-off voltages. Let's observe that the value of the real capacity of a battery depends on the definitions of the cut-off voltages. In fact, if the definitions of the EOD and EOC are different from those specified by the battery manufacturer, then also the battery maximum capacity will be different from the nominal capacity.

Once the cut-off voltages have been specified, it is possible to introduce the definition of the state of charge. For a battery, the state of charge, or SOC, is the ratio of the amount of charge left in a battery to its maximum capacity. The state of charge is usually expressed in percentage. Its value ranges in $[0\,\%, 100\,\%]$, where $\mathrm{SOC} = 0\,\%$ corresponds to the full-discharge condition of the battery and $\mathrm{SOC} = 100\,\%$ corresponds to full-charge condition.

The open-circuit voltage, or OCV, represents the voltage at the terminals of the battery when there is no load applied. The open-circuit voltage depends, in first approximation, on the battery SOC and on its internal temperature.

Figure 1 shows the trend of the OCV in respect to the SOC of the battery, at a fixed temperature, for two different categories of lithium-ion batteries.

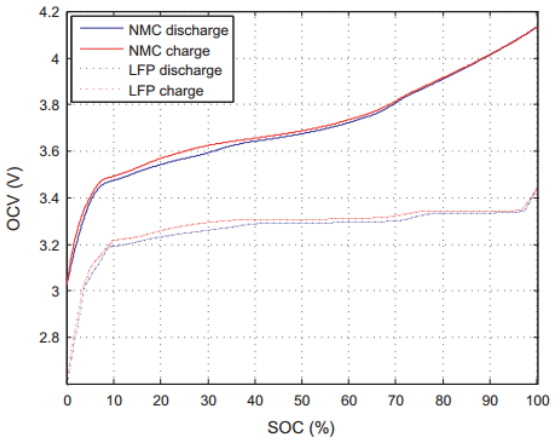

Figure 1: Example of OCV-SOC curves.

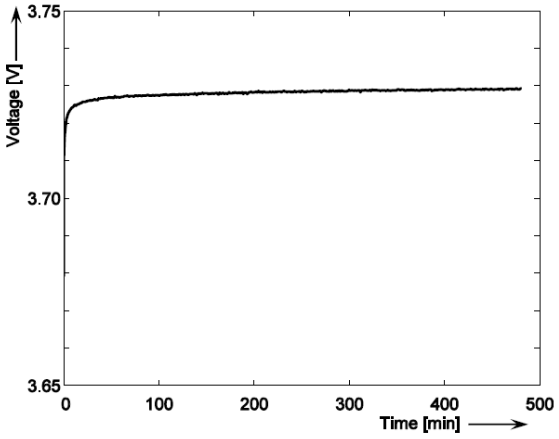

Figure 2: Relaxation process [2].

Figure 1 reveals another important aspect. In the OCV versus SOC plane, the LFP batteries exhibit the discharge curve lying well below the charge curve. This phenomenon is called hysteresis and it occurs, more or less evidently, in all types of lithium-ion batteries.

Figure 2 shows another characteristic of the lithium-ion battery behaviour. Let's observe that the value of the open-circuit voltage is not reached immediately when the battery current passes from a certain value to zero. After that the current becomes zero, in fact, the voltage at the battery terminals is subject to a very slow convergence to the corresponding value of the OCV, generally in the order of hours. This particular behaviour is due to the battery internal processes and it is called relaxation effect [2].

Let's pass to the definition of the C-rate. The C-rate is an expression describing the value of the charging and discharging current in normalized form. The general expression is "C/x", where "x" indicates the number of hours to completely discharge the battery at a constant current. So C/20 is the current at which the battery will last for 20 hours, C/1, or 1C, is the current at which the battery will last 1 hour. Hence, a value of current of 1C corresponds numerically to the battery maximum capacity expressed in Ah.

2.1.2 Li-ion battery chemistries

Lithium-ion batteries are composed by a cathode (positive electrode), an anode (negative electrode) and an electrolyte as conductor.

Unlike the lead-acid, nickel metal hydride or nickel-cadmium batteries, there is no fixed chemistry for the lithium-ion cell, but it depends on the combination of anode, cathode and electrolyte materials.

In general, the cathode is of metal oxide while the anode consists of porous carbon. During discharge, the ions flow from the anode to the cathode through the electrolyte and separator. Charge reverses the direction and the ions flow from the cathode to the anode.

Figure 3 shows the most common lithium-ion chemistries.

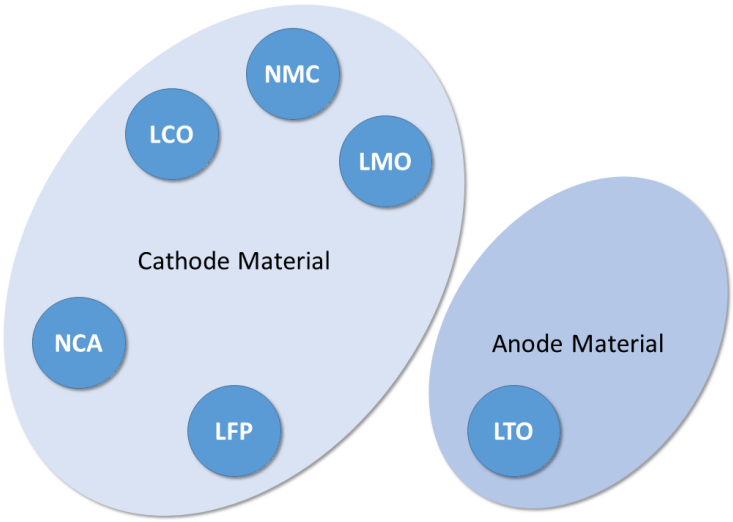

Figure 3: Different technologies of Li-ion batteries.

A number of combinations of anode and cathode materials have been evaluated since 1990. Table 1 shows the combination between the materials of the anode and cathode in the most common Li-ion batteries.

Chemistry type	**Cathode**	**Anode**
Lithium Cobalt Oxide (LCO)	$LiCoO_2$	Graphite/Hard carbon (LiC_6)
Lithium Manganese Oxide (LMO)	$LiMn_2O_4$	Graphite/Hard carbon (LiC_6)
Lithium Nickel Manganese Cobalt Oxide (NMC)	$LiNiMnCoO_2$	Graphite/Hard carbon (LiC_6)
Lithium Iron Phosphate (LFP)	$LiFePO_4$	Graphite/Hard carbon (LiC_6)
Lithium Nickel Cobalt Aluminum Oxide (NCA)	$LiNiCoAlO_2$	Graphite/Hard carbon (LiC_6)
Lithium Titanate (LTO)	$LiMn_2O_4$ /$LiNiMnCoO_2$	$Li_4Ti_5O_{12}$

Table 1: Most common lithium-ion chemistries.

A summary of the characteristics of each lithium-ion chemistries is given below.

- **Lithium Cobalt Oxide (LCO):** The LCO cell presents a high specific energy that makes it a popular choice for portable devices, such as laptops, cellular

phones and digital cameras. The battery consists of a cobalt oxide cathode and a graphite carbon anode. The cathode has a layered structure and during discharge, lithium ions move from the anode to the cathode. The flow reverses on charge. The drawback of LCO chemistry is the relatively short lifetime, low thermal stability and limited specific power. Figure 4 shows the main characteristics of the Li-cobalt cell.

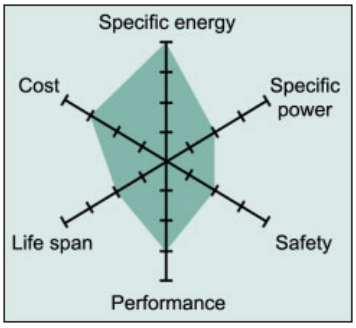

Figure 4: LCO chemistry characteristics [3].

The LCO cell present a nominal voltage of 3.6 V, while the typical operating range is between 3.0 V and 4.2 V. Moreover LCO cells cannot be charged and discharged at a current higher than its C-rate. Forcing a fast charge or applying a load higher than 1C current causes overheating and undue stress.

- **Lithium Manganese Oxide (LMO):** Li-ion with manganese spinel was first published in the Materials Research Bulletin in 1983. In 1996, Moli Energy commercialized a Li-ion cell with lithium manganese oxide as cathode material. The architecture forms a three-dimensional spinel structure that improves ion flow on the electrode, which results in lower internal resistance and improved current handling. A further advantage of spinel is high thermal stability and enhanced safety, but the cycle and calendar life are limited. Figure 5 shows the spider web of a typical Li-manganese battery.

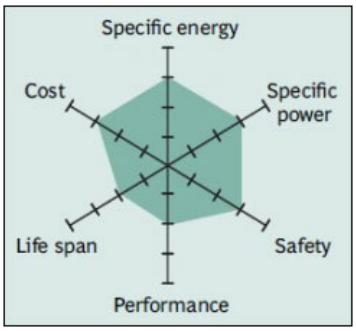

Figure 5: LMO batteries characteristics [3].

Li-manganese cell exhibits a typical nominal voltage of about 3.7 V. The typical operating range is between 3.0 V and 4.2 V. Low internal cell resistance enables fast charging and high-current discharging. Li-manganese is used for power tools, medical instruments, as well as hybrid and electric vehicles.

- **Lithium Nickel Manganese Cobalt Oxide (NMC):** NMC is one of the most successful Li-ion technology. It presents a nickel-manganese-cobalt cathode. The secret of this chemistry lies in combining nickel and manganese. Nickel is known for its high specific energy but poor stability. Manganese has the benefit of forming a spinel structure to achieve low internal resistance but offers a low specific energy. Combining the metals enhances each other strengths. Figure 6 shows the main characteristics of a typical Lithium Nickel Manganese Cobalt Oxide battery.

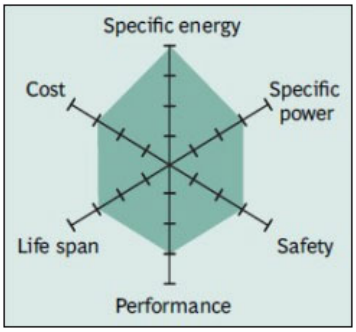

Figure 6: NMC chemistry main characteristics [3].

The typical operating range is the same shown for the LCO and LMO cells while the typical nominal voltage lies in the range between 3.6 V and 3.7 V. NMC has good overall performance and excels on specific energy. This battery is the preferred candidate for the electric vehicle and has the lowest self-heating rate.

- **Lithium Iron Phosphate (LFP):** LFP has been discovered as cathode material for rechargeable lithium batteries in 1996, by the University of Texas. It offers good electrochemical performance with low resistance. The key benefits are high current rating and long cycle life, besides good thermal stability, enhanced safety and tolerance if abused. Figure 7 shows the characteristics of this chemistry.

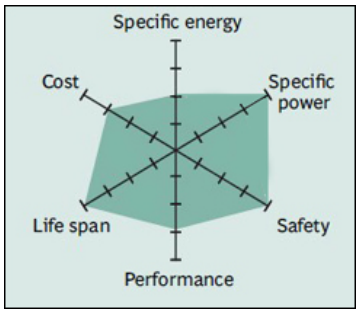

Figure 7: LFP technology characteristics [3].

The nominal voltage of the LFP cell lies between 3.2 V and 3.3 V. The typical operating voltage range is [2.5 V, 3.65 V]. LFP has excellent safety and long life span but moderate specific energy and elevated self-discharge. As in most batteries, cold temperature reduces performance and elevated storage temperature shortens the service life, and LFP is no exception. Moreover, the LFP cell exhibits a marked hysteresis.

- **Lithium Nickel Cobalt Aluminum Oxide (NCA):** Lithium Nickel Cobalt Aluminum Oxide battery, or NCA, has been introduced around since 1999 for special applications. This chemistry shares similarity with NMC by offering high specific energy and reasonably good specific power and a long life span. Less good are safety and cost. NCA is a further development of lithium nickel oxide. Adding aluminium gives the chemistry greater stability. Figure 8 shows the spider web of a typical Lithium Nickel Cobalt Aluminum Oxide battery.

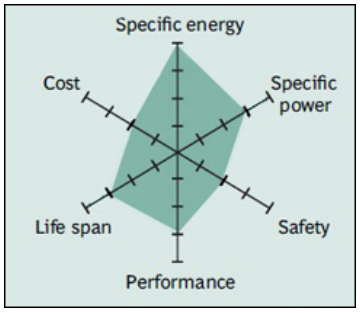

Figure 8: NCA chemistry characteristics [3].

The nominal voltage is of 3.6 V, while the typical operating range is [3.0 V, 4.2 V]. High energy and power densities, as well as good life span, make NCA a candidate for EV powertrains. High cost and marginal safety are negatives.

- **Lithium Titanate (LTO):** Cells with lithium titanate anodes have been known since the 1980s. Li-titanate replaces the graphite in the anode of a typical lithium-ion battery and the material forms into a spinel structure. The cathode is graphite

and resembles the architecture of a typical lithium-metal battery. LTO has a nominal cell voltage of 2.4 V, can be fast charged and delivers a high discharge current of 10C, or 10 times the rated capacity. The cycle count is said to be higher than that of a regular Li-ion. Li-titanate is safe and has excellent low-temperature discharge characteristics. LTO cell charges to 2.8 V and the end of discharge is 1.8 V. Figure 9 illustrates the characteristics of the Li-titanate battery.

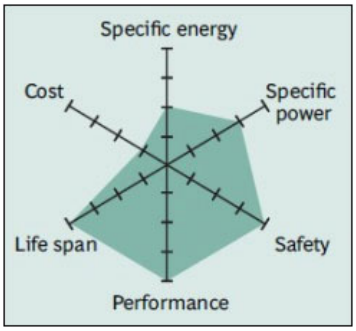

Figure 9: LTO chemistry main characteristics [3].

Li-titanate excels in safety, low-temperature performance and life span. Efforts are being made to improve the specific energy and lower cost. Typical uses are electric powertrains and solar-powered street lighting.

Figure 10 shows a comparison between the various technology of Li-ion batteries and the lead–acid, nickel–cadmium (NiCd) and nickel–metal hydride (NiMH) batteries. The comparison is done observing the typical specific energy of each chemistry.

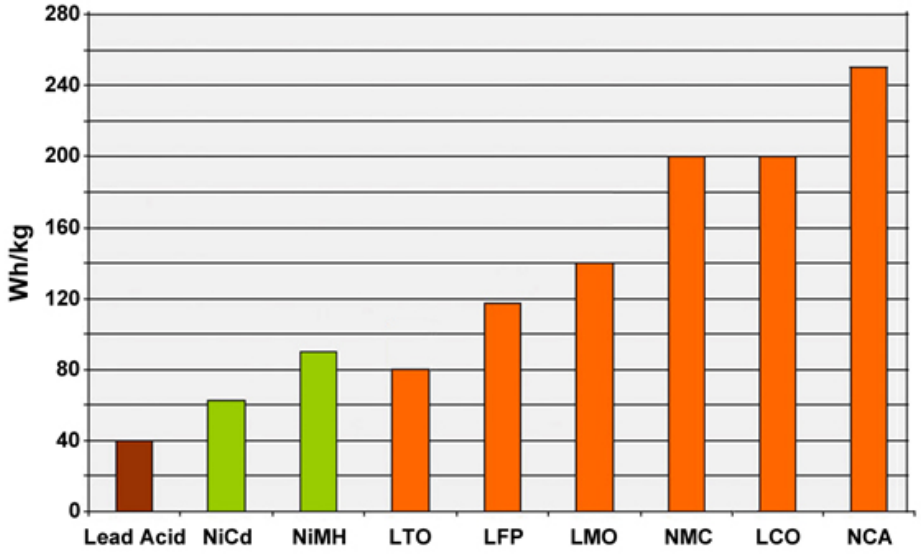

Figure 10: Comparison of battery chemistries in relation to their specific energy [3].

NCA enjoys the highest specific energy. However, manganese and phosphate are superior in terms of specific power and thermal stability. Li-titanate has the best life span.

2.2 Li-ion battery modelling

2.2.1 Battery model types

As introduced in section 1, the aim of this work is the development of a behavioural model of a Li-ion battery. The model has to account for the electrical behaviour of the battery, i.e., given the battery current, the model should be able to predict the battery internal state, expressed with the state of charge, and the battery voltage at the external terminals.

In literature, a great number of battery models have been presented. While detailed physics-based models have been built to study the internal dynamics of the batteries, these models are generally not suitable for system-level design exercises [4]. On the other hand, simple dynamic models based on capacitor and resistor networks that can be used in a circuit simulator are generally so simplified that they lack interesting features such as non-linear phenomena description and temperature effects. Although non-linear phenomena can be included in circuit-based models, it dramatically increases the complexity of the modelling process.

The battery models can be generally divided into several categories which differ in the the level of abstraction they assume in respect with the physical nature of the battery.

Figure 11 shows the characteristics of the main categories of battery models, in relation to their predictability and their intrinsic complexity, expressed as CPU time in simulations [5].

The battery models can be generally organized in three different categories, electrochemical models, mathematical models and empirical models. Each category of battery models in turn contains several model types, that are different for their characteristics and complexity.

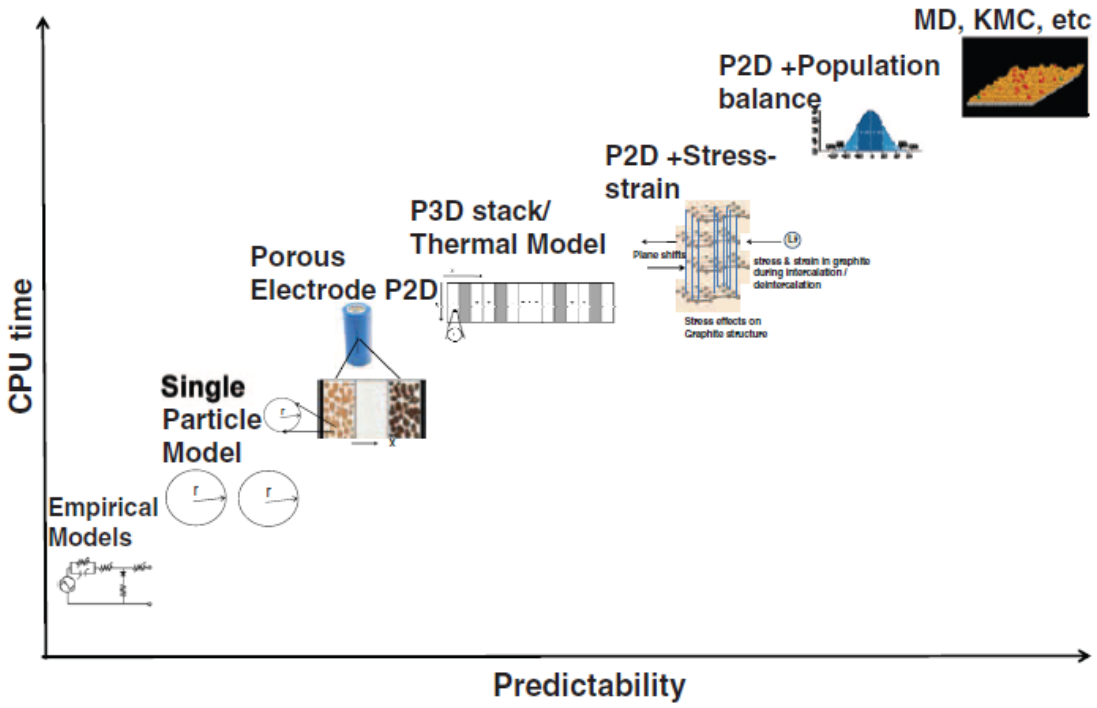

Figure 11: Different categories of battery models [5].

The electrochemical models are based on the description of the electrochemical processes that take place inside the battery [6]. The models describe these processes in great detail. To do this, they use complex equations accounting for the microscopic and macroscopic behaviours of the battery. This makes these models the most accurate. However, these models are the most complex and its simulations are very time-consuming because they involve a system of coupled time variant spatial partial differential equations. In addition, the electrochemical models require some specific information about the battery behaviour that is difficult to obtain, because of the proprietary nature of the technology. To obtain the values of all these parameters, a very detailed knowledge of the battery to be modelled is required. For this reason, electrochemical models, in general, are not used for system-level simulations aimed at the design of battery management systems or chargers, but rather they are mainly used to optimize the physical design aspects of the batteries.

Another category of battery models is represented by the mathematical models. They present this name because adopt empirical equations or mathematical methods like stochastic approaches to predict the battery behaviour. They are often too abstract to embody any practical meaning but they can be still useful to system designers. The main reason for their low utilization is that the mathematical models cannot offer any information about the values of the current and of the voltage of the battery, that are necessary for the design and for the simulation and optimization of the circuits interfacing with the battery.

For this reason, the empirical models, are preferred in respect to the mathematical. The empirical models are represented basically by equivalent circuit models (ECM). Equivalent circuit models are composed by a combination of voltage and current sources, resistors, and capacitors in order to be used in the design and in the simulation with other electrical circuits and systems. Their accuracy lies between that of the electrochemical models and that of the mathematical models. They are often used because they are more intuitive, useful, and easy to handle for electrical and electronic engineers, especially when they are used in circuit simulators and alongside with application circuits.

So, in order to be easily used in simulation, the battery model has to be implemented as an electrical equivalent model.

2.2.2 Equivalent circuit models

In literature, between the equivalent circuit models, we can find a lot of different circuits that model the battery electrical behaviour. The models proposed in this section have been described in [7].

Three different examples of equivalent circuit models are shown in figure 12.

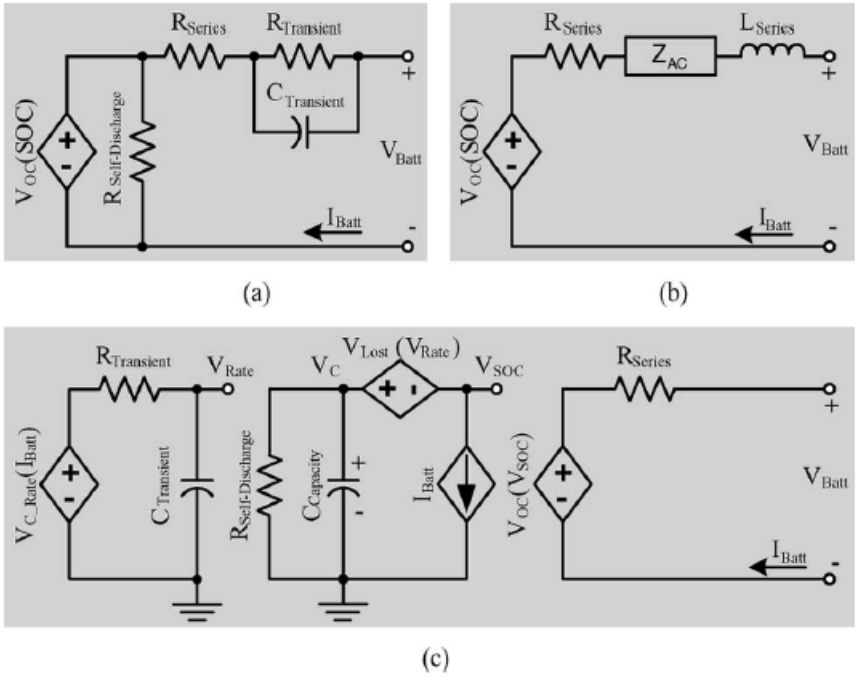

Figure 12: Examples of equivalent circuit models [7].

In its most basic form, a Thevenin-based model, represented in figure 12(a), uses a series resistor and an R-C parallel network to predict the transient response of the battery. The model is able to work only at a particular SOC, by assuming the OCV

constant. This assumption is extremely limiting since that it prevents the model from capturing steady-state battery voltage variations, i.e., the dc response, as well as runtime information. Some derivative models has been introduced to add the improvements necessary to predict runtime and dc response, but they still have several disadvantages. For example, a variable capacitor can be used instead of OCV(SOC) to represent the non-linear relationship between open-circuit voltage and state of charge, but it complicates the capacitor parameter. Some models represent only the non-linear relation between the OCV and the SOC, but ignores the transient behaviour. Sometimes additional mathematical equations are implemented to obtain the SOC and estimate runtime, but they cannot be implemented in circuit simulators. Other models adopts R-C parallel networks with two time-constants, but only works at a particular SOC and temperature condition. Other models, finally, employ a complicated electrical network extracted from physical process to model open-circuit voltage, which complicates the whole model. Thus, none of these Thevenin-based models can predict the battery runtime simply and accurately in circuit simulators.

Impedance-based models, an example of which is shown in figure 12(b), employ the method of electrochemical impedance spectroscopy to obtain an ac-equivalent impedance model in the frequency domain, and then use a complicated equivalent network to fit the impedance spectra [8]. The fitting process is difficult, complex, and non intuitive. In addition, impedance-based models only work for a fixed SOC and temperature setting, and therefore they cannot predict dc response or battery runtime.

Runtime-based models, shown in figure 12(c), use a complex circuit network to simulate battery runtime and dc voltage response for a constant discharge current in SPICE-compatible simulators. Some derivative models are continuous-time implementations in SPICE simulators and discrete-time implementations using VHDL language. They can predict neither runtime nor voltage response for varying load currents accurately.

So, none of these models can be implemented in circuit simulators to predict both the battery runtime and I–V performance accurately.

2.2.3 Reference battery model

Among the different battery equivalent circuit models, the model chosen to represent the behaviour of the battery is that shown in the figure 13.

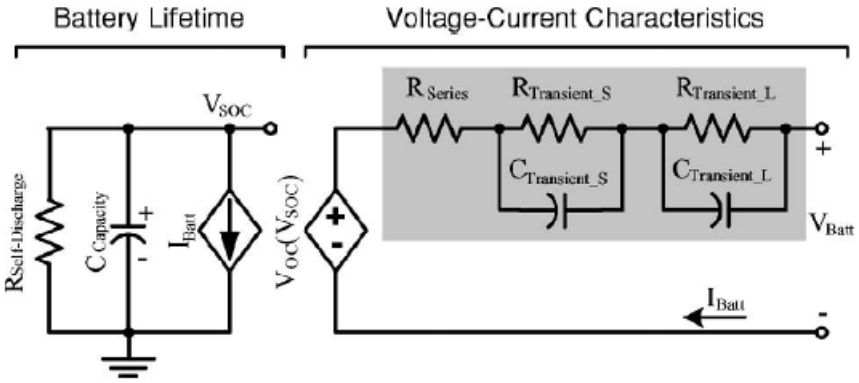

Figure 13: Equivalent circuit model of reference [7].

Also this model has been presented in [7], as an accurate, intuitive, and comprehensive electrical battery model, able to overcome the defects of the battery models presented so far. Let's describe the characteristics of the ECM.

The left-hand side section of the model represents the battery lifetime.

- $R_{self-discharge}$ models the self-discharge phenomenon of the battery. This effect is only observable when a battery is left idle or disconnected for a relatively long time. As a result, the self-discharge affects mainly the batteries used for power applications like electric or hybrid vehicles and in general for automotive applications. In this work the contribution of this phenomenon is neglected. In mobile applications, in fact, the battery is continuously discharged and recharged and practically is never left idle. Consequently, the effect of self-discharge cannot be observed and so it can be reasonably neglected.

- $C_{capacity}$ models the capacity of the battery. The voltage on this capacitor, V_{soc}, represents the battery state of charge. In fact, since the value of $C_{capacity}$ is computed as the cell maximum capacity divided by 1 V, the voltage V_{soc} ranges from 0 to 1. In particular, if $V_{soc} = 1$, then the battery is considered fully charged (SOC = 100 %), while if $V_{soc} = 0$, then the battery is considered fully discharged (SOC = 0 %).

- $I_{battery}$ is a current-controlled current source that is driven by the battery charging current. The main purpose of the generator is to decouple the section of the circuit that model the battery lifetime by the output section that represents the voltage-current characteristics. This current, flows into the capacitor $C_{capacity}$

and this modifies the battery SOC. So at a generic instant t, the SOC can be computed, according to the Coulomb counting method, as:

$$SOC(t) = SOC(t_0) + \frac{100}{C_{capacity}} \int_{t_0}^{t} I_{battery}(\tau) d\tau \tag{1}$$

indicating with $SOC(t_0)$ the SOC of the battery at the instant t_0.

The right-hand section of the model represents the output characteristics of the battery, expressed a relationship between the voltage and the current of the battery.

- $V_{open-circuit}$ is a voltage-controlled voltage source used to bridge the SOC to the OCV. The generator is driven by the voltage V_{soc}, that expresses the SOC of the battery and models the non-linear relationship between the OCV of the battery and its SOC.

- R_{series} represents the ohmic resistance of the battery. Considering the electrochemical aspects, this resistance is related to transport of the lithium ions inside the electrolyte solution.

- The R-C groups model the behaviour of the battery in the transients. In literature, model with one, two, or an higher number of R-C groups are commonly presented. In general, the more the number of R-C groups, the more the accuracy and the the complexity of the model. In this case, a circuit with two R-C branches has been chosen to represent the battery behaviour in order to have a compromise between accuracy and complexity.

  $R_{transient-short}$ and $C_{transient-short}$ are introduced to characterize the short term processes of the battery transient response. From the electrochemical points of view, they are related to the charge transfer and to the double-layer capacitance.

  $R_{transient-long}$ and $C_{transient-long}$, instead, model the long term processes of the battery transient response and they are related to the mass transport and to the diffusion process.

- Finally $V_{battery}$ is the output voltage, considered at the battery external terminals.

The electrochemical behaviour of the battery depends on state of charge, temperature and current rate. So, the battery parameters can be written, expliciting their dependencies, as:

$$V_{open-circuit} = V_{open-circuit}(SOC, I_{battery}, T) \tag{2}$$
$$R_{series} = R_{series}(SOC, I_{battery}, T) \tag{3}$$
$$R_{transient-short} = R_{transient-short}(SOC, I_{battery}, T) \tag{4}$$
$$C_{transient-short} = C_{transient-short}(SOC, I_{battery}, T) \tag{5}$$
$$R_{transient-long} = R_{transient-long}(SOC, I_{battery}, T) \tag{6}$$
$$C_{transient-long} = C_{transient-long}(SOC, I_{battery}, T) \tag{7}$$

2.3 Simulation Environments

2.3.1 Preliminary description

The purpose of this work is the development of a behavioural model of a lithium battery. The battery model needs to be realized into different environments in order to be used for the design and the optimization of battery management systems and battery charging in high-level, analog and digital point of view. For this reason, three different models have been realized in three different environments. All the models are based of the ECM presented in the previous section. The first model is realized in a high-level environment such as Matlab/Simulink, the second in an analog environment and the third in a digital environment.

In the next sections, a brief introduction to the different implementation and simulation environments used for the development of the battery models is presented.

2.3.2 Matlab/Simulink

The first battery model has been implemented as an high-level model and it is realized in the Matlab/Simulink environment. A representation of the environment is shown in figure 14.

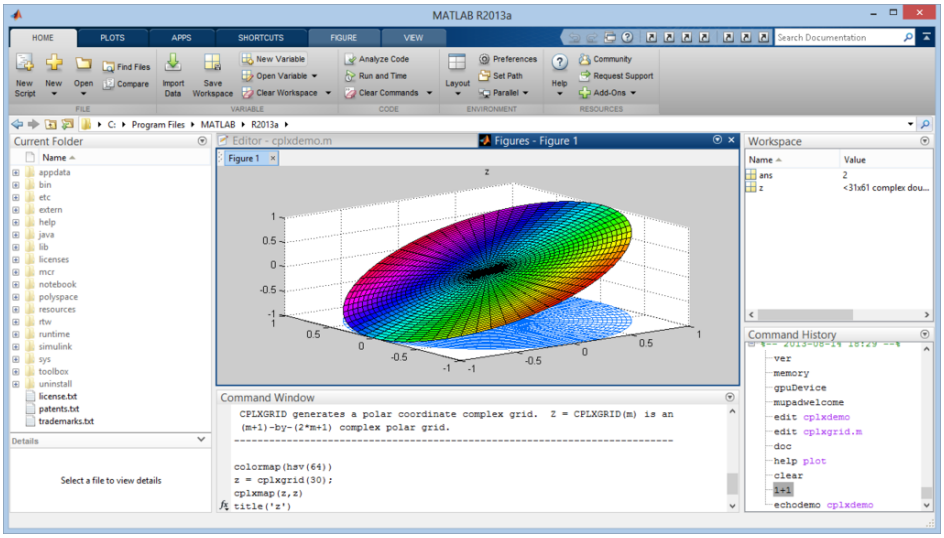

Figure 14: Matlab environment.

Matlab (matrix laboratory) is a multi-paradigm numerical computing environment and programming language, developed by MathWorks [9]. Matlab allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces and interfacing with programs written in other languages, including C, C++, Java, Fortran and Python.

Simulink, again developed by MathWorks, is an additional package that adds a graphical programming environment for modelling, simulating and analysing multi-

domain dynamic systems [10]. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries. It offers tight integration with the rest of the Matlab environment and can either drive Matlab or be driven from it. Simulink is widely used in automatic control and digital signal processing for multi-domain simulation and model-based design.

2.3.3 Cadence Virtuoso

The second battery model has been implemented as analog model. The environment in which the model has been realized is Cadence Virtuoso. The Cadence Virtuoso platform is represented by a set of tools for designing full-custom integrated circuits. It includes schematic entry, behavioural modelling, circuit simulation, custom layout, physical verification, extraction and back-annotation [11]. The Cadence Virtuoso platform is mainly used for analog, mixed-signal, RF, and standard-cell designs, but also for memory and FPGA designs. Figure 15 shows the Virtuoso environment.

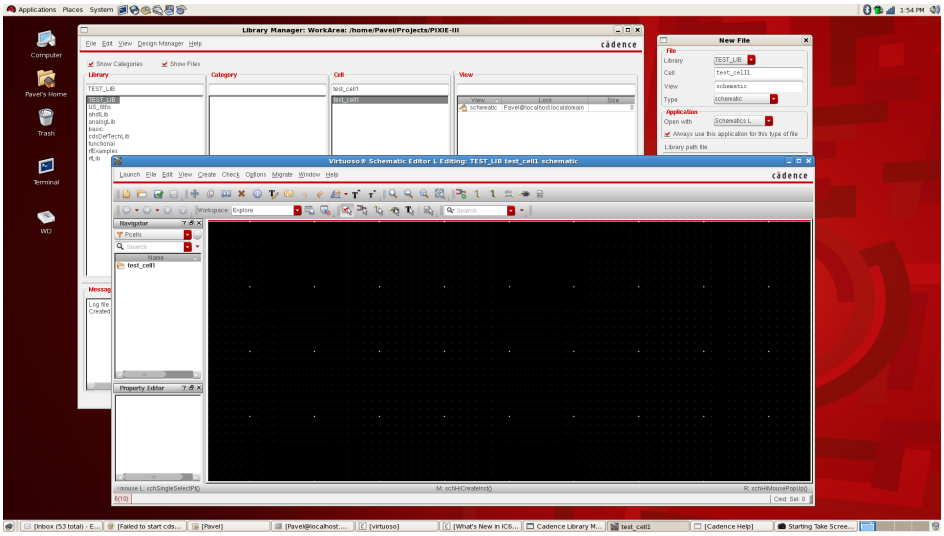

Figure 15: Cadence Virtuoso environment.

Once the battery model has been implemented using the Schematic Editor provided by the Virtuoso platform, it can be simulated using Spectre Circuit Simulator. Spectre is an advanced circuit simulator that simulates analog and digital circuits at the differential equation level. Besides the basic capabilities, the Spectre circuit simulator provides significant additional capabilities over SPICE. SpectreHDL (Spectre High-Level Description Language) and Verilog-A use functional description text files to model the behaviour of electrical circuits and other systems.

After the battery model simulation, the simulation results can be displayed using the Visualization and Analysis Tool in order to verify and debug the design.

2.3.4 Cadence Incisive

The third battery model is implemented in a digital environment. The implementation is done using SystemVerilog HDL. In the semiconductor and electronic design industry, SystemVerilog is a combined hardware description language and hardware verification language, based on extensions to Verilog. It can be used for RTL design as an extension of Verilog-2005, even though its main goal is the verification [12].

The use of SystemVerilog language for the description of the battery model allows you to simulate the model in a discrete-event simulator. A discrete-event simulation models the operation of a system as a discrete sequence of events in time. Each event occurs at a particular instant in time and marks a change of state in the system. Between consecutive events, no change in the system is assumed to occur and thus the simulation can directly jump in time from one event to the next. This contrasts with continuous simulation in which the simulation continuously tracks the system dynamics over time. Instead of being event-based, this is called an activity-based simulation. In this case, the time is broken up into small time slices and the system state is updated according to the set of activities happening in the time slice. Because discrete-event simulations do not have to simulate every time slice, they can typically run much faster than the corresponding continuous simulation. So, once the model has been described in SystemVerilog, it can be simulated using a discrete-event simulator. The simulation of the model is performed in the Cadence Incisive Enterprise Simulator environment. Figure 16 shows the environment aspect.

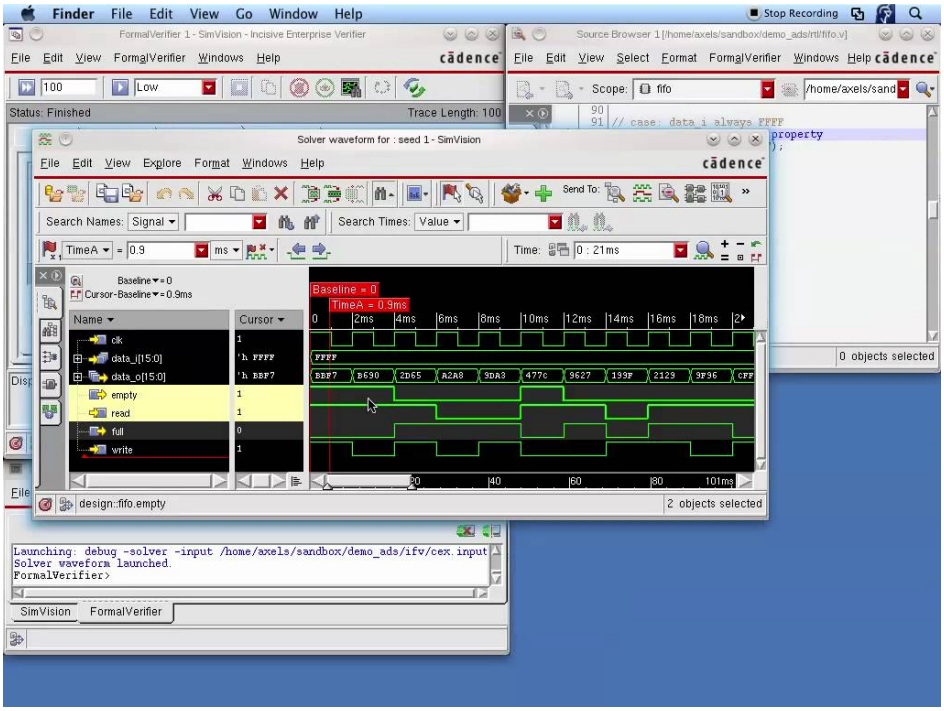

Figure 16: Cadence Incisive environment.

Incisive is a suite of tools from Cadence Design Systems related to the design and verification of ASICs, SoCs, and FPGAs [13]. It is commonly referred to by the name NCSim in reference to the core simulation engine. Incisive provides also a unified graphical debugging environment called SimVision Debug. It can be used to debug digital, analog, or mixed-signal designs written in Verilog, SystemVerilog, VHDL, and SystemC languages or in a combination thereof.

3 Battery characterization

3.1 Experimental setup

In this section, the battery characterization procedure is presented. As introduced in section 1, the battery to be modelled is the one that powers the Samsung Galaxy Note 4 smartphone. The battery is lithium-ion and belongs to the NMC chemistry, in which the cathode is composed by a combination of nickel-manganese-cobalt. The battery presents a nominal capacity of 3220 mA h and an end-of-charge voltage of 4.4 V.

To extract the parameters of the equivalent circuit model of the battery, the battery has to be firstly characterized with a well defined experiment. The test used for the battery characterization is called pulsed current test (PCT) and will be presented in detail in the following.

The pulsed current test is carried out using the experimental setup shown in the figure 17.

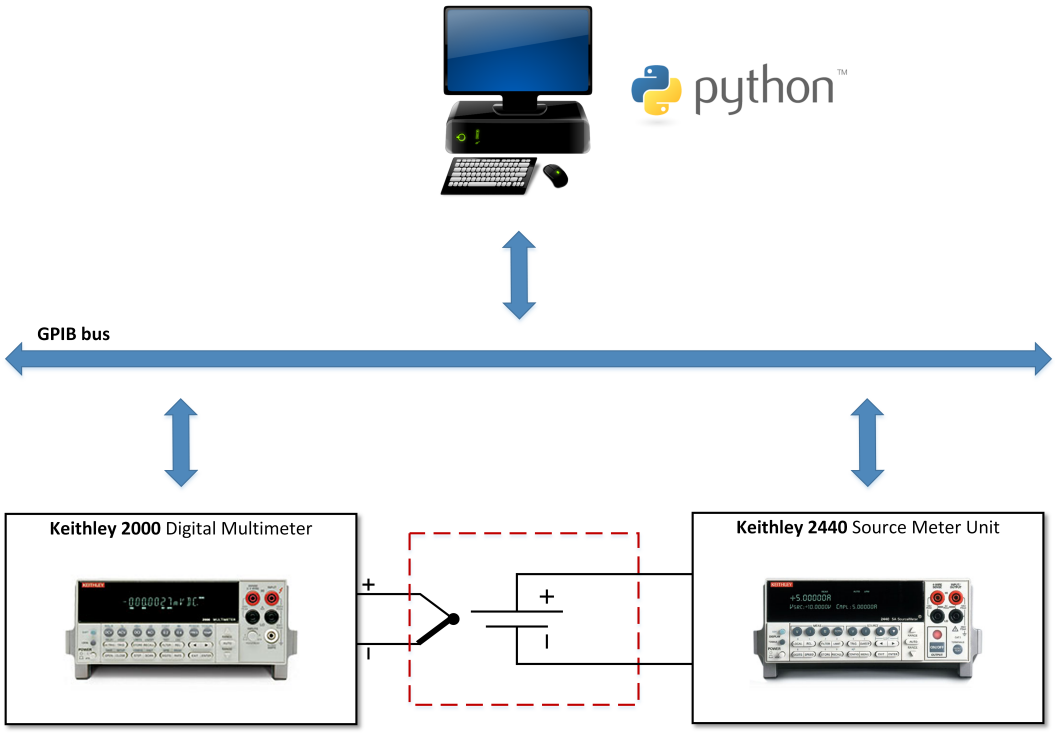

Figure 17: Experimental setup representation.

The instruments used for the battery characterization are the `Keithley 2440` and the `Keithley 2000`.

The `Keithley 2440` is a source-meter unit able to provide four-quadrant operations. It can deliver or absorb current up to 5 A and can set or measure voltages up to 40 V, up to a maximum output power of 50 W. It can also be controlled in remote operation from a host computer using the RS-232 or the GPIB interface. The `Keithley 2440`

is used during the test both to drive the battery and to acquire measurements. The battery can be driven with a current or with a voltage. If the battery is driven with a current, then the instrument has to be configured to set the current and measure the voltage, while if the battery is driven with a voltage, then the instrument has to be configured to set the voltage and measure the current. In order to reduce the contribution of the parasitic resistances of the leads, the battery is connected to the instrument with a four-wire configuration.

The `Keithley 2000` is a digital multimeter. It is used during the test only to monitor the cell temperature. The instrument is in fact equipped with a functionality that allows you to measure the temperature in the case where a thermocouple is connected to its input. To correctly measure the temperature, the thermocouple type has to be specified in the instrument settings. So for this measurement, a K type thermocouple from Fluke is used. Like the other instrument, also the operations performed by the `Keithley 2000` can be controlled from a host computed using the RS-232 or the GPIB interface.

As shown in the figure 17, the instruments are connected together and to the host computer via a GPIB communication bus. During the pulsed current test, in fact, the acquisition rate of the voltage, current and temperature samples is of 10 Hz and the GPIB interface allows you to have better performance than the RS-232.

The communication between the instruments and the host computer is realized using Python language. Python is a widely used general-purpose, high-level programming language. It is interpreted and object-oriented and in the last years has gained popularity because of its clear syntax and of its readability. Its simple syntax allows programmers to express concepts in a fewer number of lines of code than would be possible in languages such as C++ or Java, making the code more clear and understandable [14]. Python supports modules and packages, which encourages program modularity and code reuse. In addition to large and comprehensive standard libraries, it provides additional modules with a wide variety of purposes: web and internet development, network programming, scientific and numeric applications, education and so on. Once one of these libraries has been imported into a Python program, functions defined within the library are automatically made available to the user.

Specifically for automating the test procedures, Python language provides a specific library named PyVisa. This library can be imported into a program to allow the computer running Python interpreter to interface with laboratory instruments, connected to the host computer with a certain communication protocol, according to the VISA standard. In this way it is possible to launch a routine that drives the instruments, takes the measurements, acquires the results and finally saves them in an output file. All the operation are performed automatically. This is very useful when deal with batteries, since the tests that involve batteries are very long and time-expensive.

Python is a very versatile language and allows users to modify and personalize the standard libraries. So, the PyVisa library can be encapsulated in modules of higher abstraction level. This can be useful to create special modules for interfacing with specific instruments, for example the `Keithley 2440` and the `Keithley 2000`, used

for the characterization of the battery.

Modules of this type have been provided by Dialog Semiconductor. The first is called `Keithley2400.py` and defines a set of functions to interact with the `Keithley 2400` series of source-meter units, between which lies the `Keithley 2440`. Among all the functions defined in this library, only a small part of them have been really used during the test. These functions are reported below.

- `set_v(`*`value`*`)` and `set_i(`*`value`*`)` set respectively the output voltage and the output current of the `Keithley 2440` at the value expressed by the argument *value*.
- `set_v_range(`*`value, mode`*`)` and `set_i_range(`*`value, mode`*`)` set the voltage and the current measurement range respectively at the value specified by *value*. These functions also allow you to specify, using the argument *mode*, if the instrument has to be configured for sourcing (*mode* = *source*) or if it has to be configured only to measure (*mode* = *sense*).
- `set_v_limit(`*`limit`*`)` and `set_i_limit(`*`limit`*`)` allow you to specify a compliance value *limit* respectively for the voltage and for current that the instrument can set or measure.
- `read_v()` and `read_i()` read respectively the voltage and the current of the `Keithley 2440` and return the value of the measurement just acquired.

Also for the `Keithley 2000`, a specific library has been provided. The library name is `Keithley2000.py`. As previously mentioned, the purpose of this instrument is to read the temperature measured by the thermocouple. So, the only function of the library `Keithley2000.py` used during the test is `read_temp()`.

- `read_temp()`, like `read_v()` and `read_i()`, reads the value of the temperature measured by the thermocouple and returns it.

As mentioned above, the Python libraries `Keithley2400.py` and `Keithley2000.py` are property of Dialog Semiconductor. The use of these libraries has been very useful and has speeded up writing of the Python scripts that control instruments during the tests.

As introduced in the section 2, the battery cell to be characterized is that relative to the Samsung Galaxy Note 4 smartphone. The nominal capacity of this cell is of 3220 mA h. For the connection between the battery and the instruments, a custom holder has been realized. The battery holder is shown in the figure 18.

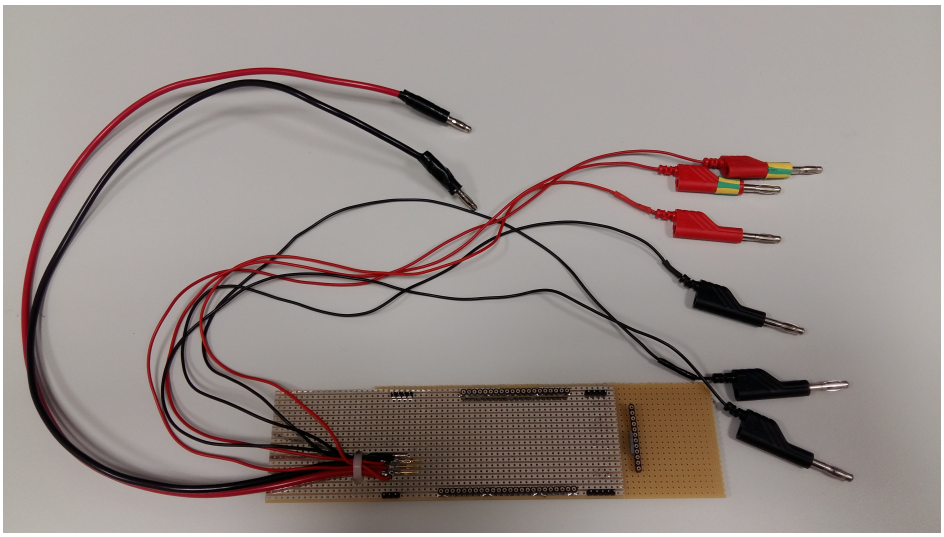

Figure 18: Custom battery holder.

The battery holder is fully customized and has been produced specifically for the battery to be characterized. It is equipped with pressure contacts which allow to embed the battery during the test. The disposition of the battery pins is, from the external to the internal, `V+`, `T`, `V-` and finally `ID`. `V+` and `V-` represent respectively the positive and negative battery terminals, `T` is a control pin at which a thermistor is connected, used to monitor the battery temperature, `ID` represents to a battery identifier. The only two contacts used in the test are `V+` and `V-`.

The thermocouple is instead placed in contact with the body of the battery using an insulating tape. In this way it is possible only to measure the temperature of the battery on the exterior case, but this is enough for the purpose of this work. The thermocouple is equipped with a specific connector that allows to connect it directly to the `Keithley 2000` and so, in this case, nothing of custom has to be made.

Once the battery holder has been realized, the experimental setup is complete and ready to perform the test. The figure 19 shows the final experimental setup.

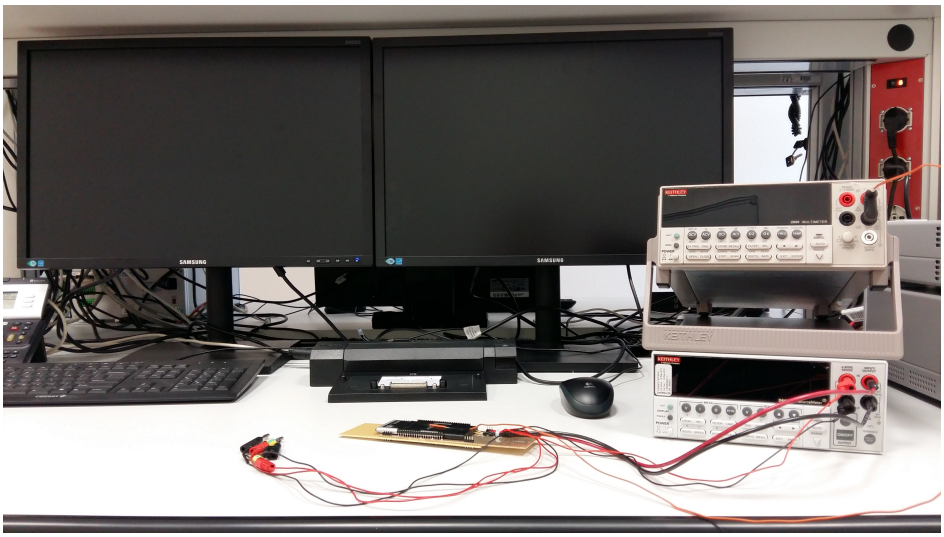

Figure 19: Experimental setup.

3.2 Initial battery training

Once the realization of the holder has been completed, the battery should be subjected to a sort of initial training made up of ten cycles of full discharge and full charge after the delivery by the manufacturer. The cell capacity, in fact, is not constant in the early cycles of use after the production and the initial cycles of discharge and charge have the task of overcoming this first phase making the capacity of the cell stable for the pulsed current test.

In the following, the initial training procedure is presented in detail. Before that, it is necessary to introduce the charging mode of a Li-ion battery and the definitions of the limits of the state of charge. A single Li-ion cell is charged in two different phases: constant-current (CC) and constant-voltage (CV). During the CC phase, a constant current is applied to the battery. So, the battery voltage steadily increases, until its value reaches the EOC voltage. During the CV phase, a voltage equal to the EOC value is applied to the battery. As a result, in this phase, the current gradually declines towards zero. During the tests, a CC-CV method is used not only for charging the battery, but also for discharging it.

Let's introduce the definitions of the limits of the state of charge. The SOC is 100 %, and so the battery is considered fully charged, when, once it is charged at an EOC voltage of 4.35 V using a current rate of C/2, the current amplitude reaches the value of C/20 during the constant-voltage phase. Similarly, the SOC is 0 %, and so the battery is considered fully discharged, when, once it is discharged at an EOD voltage of 2.90 V using a current rate of C/2, the current amplitude reaches the value of C/20 during the constant-voltage phase.

Let's return to the description of the initial training procedure. As introduced previously, the procedure is described using Python language. The data acquisition

rate is of 0.2 Hz in order that the log files produced during the test are not too large. The test is realized at room temperature. The data acquisition has been implemented so that the tests are carried out safely. The python program is in fact equipped with security measures that allow you to disconnect the battery from the `Keithley 2440` source-meter in case of over-voltage, under-voltage, over-current and over-temperature and so to end the test.

The test parameters used in the Python program are listed below.

```
# Test parameters.
V_LIMIT_UP = 4.4          # Maximum acceptable value of the voltage.
V_LIMIT_LOW = 2.85        # Minimum acceptable value of the voltage.
I_LIMIT = 1.8             # Maximum acceptable value of the current.
TEMP_MAX = 30.0           # Maximum acceptable value of the temperature.
V_EOC = 4.35              # End of charge voltage value to set in charging.
V_EOD = 2.9               # End of discharge voltage value to set in discharging.
I_1C = 3.22               # 1C rate current.
I_COMPL = I_1C / 2        # Current compliance in charging and discharging.
I_EOCV = I_1C / 20        # End of Costant Voltage phase current.
V_RANGE = 10.0            # Voltage measurement range.
I_RANGE = 5.0             # Current measurement range.
T_REST = 3600.0           # Resting period duration.
```

The Python program that control the acquisition from the instrument is made up of four successive phase:

- Firstly, the battery is fully discharged with a CC-CV method. In this phase the `Keithley 2440` is set as a voltage source with a current compliance. The voltage value is set to 2.90 V, that corresponds to the EOD voltage, while the current limit is set to 1.61 A, corresponding a to current rate of C/2. In this mode, the instrument is able to realize both the constant-current and the constant-voltage phase. Consider that the battery voltage at the beginning of this phase is higher than 2.90 V. Since that, initially the instrument would absorb a current higher than the limit. However, the current is limited to a rate of C/2. So the current limit is reached and then the instrument behaves as a current source. At this point the voltage of the battery gets down until it reaches the EOD voltage. Once the voltage reaches the EOD voltage, the value of the current absorbed by the instrument return into the limits and then the instrument switches to the constant-voltage phase, forcing a voltage of 2.90 V. The current gradually gets down until it reached the value if 161 mA, corresponding to a current rate of C/20. At this point the SOC of the battery is considered at 0 % and the discharging phase is considered ended.

  The section of the Python script in which the `Keithley 2440` is configured to realize the discharging phase is shown below.

```
##########################################################################
# Discharging phase:
#   - Vsmu < Vbattery;
#   - Ismu < 0.
##########################################################################

# Set the SMU as voltage source with current limit.
```

```
smu.set_v(V_EOD)
smu.set_v_range(V_RANGE, mode = 'source')
smu.set_i_limit(I_COMPL)
smu.set_i_range(I_RANGE, mode = 'sense')
```

- Then a rest period of one hour is applied, in which the current is set to zero. In this period, the battery shows its relaxation phenomenon and the voltage gets up from 2.90 V to about 3.3 V, value that reflects the OCV at 0 % of the SOC.

  Below the section in which the `Keithley 2440` is configured for the rest phase is reported.

```
##############################################################################
# One hour pause:
#   - Vsmu = Vbattery;
#   - Ismu = 0.
##############################################################################

# Set the SMU as current source.
smu.set_i(0)
smu.set_i_range(0.00001, mode = 'source')
smu.set_v_range(V_RANGE, mode = 'sense')
```

- Then the battery is fully charged with a CC-CV method. As in the discharging phase, also in this the `Keithley 2440` is set as a voltage source with a current compliance. The value of the current limit is the same of that used in discharge, 1.61 A, that correspond to a current rate of C/2, while the value of the voltage is set to 4.35 V, corresponding to the EOC voltage. In this mode, the instrument initially behaves as a current source of 1.61 A and then the voltage of the cell begins to rise. The voltage rises until it reaches the value of 4.35 V. Once the EOC voltage has been reached, the instrument begins to impose that voltage on the battery terminals and then the constant-voltage phase starts. The amplitude of the current slowly gets down until it reaches the value of 161 mA, corresponding to C/20. At this point, the battery SOC is considered at 100 % and the charging phase is finished.

  The section in which the `Keithley 2440` is configured for the charging phase is reported below.

```
##############################################################################
# Charging phase:
#   - Vsmu > Vbattery;
#   - Ismu > 0.
##############################################################################

# Set the SMU as voltage source with current limit.
smu.set_v(V_EOC)
smu.set_v_range(V_RANGE, mode = 'source')
smu.set_i_limit(I_COMPL)
smu.set_i_range(I_RANGE, mode = 'sense')
```

- Finally, a new rest period of one hour is applied in which the current is again set to zero. The voltage in this case gets down from the EOC value to about 4.3 V

because of the relaxation process. This value of the voltage corresponds to the OCV value at 100 % of the SOC. The setting the `Keithley 2440` for this phase is the same shown in the first relaxation period.

As previously introduced the data acquisition rate is 0.2 Hz during the whole test. At each sampling time a well defined set of operations is performed. First of all the measurements of current and voltage are acquired by the `Keithley 2440` and transmitted to the host computer. Then also the temperature is read using the thermocouple connected to the `Keithley 2000` and sent to the computer. At this point the samples of the time, voltage, current and temperature are displayed on the computer screen using the interface provided by the Python interpreter, just to check that the acquisition is functioning properly. The same samples are then saved in an output file that records the entire test. After that the safety test is performed. It is controlled that voltage, current and temperature are within the maximum limits defined in the initial part of the Python script. Then the value of the battery current is checked during discharging and charging in order to monitor if the current reaches the value of C/20, while during the pauses, the time is controlled to check if an hour is passed.

Figure 20 shows the results of the last of the ten cycles of the initial training.

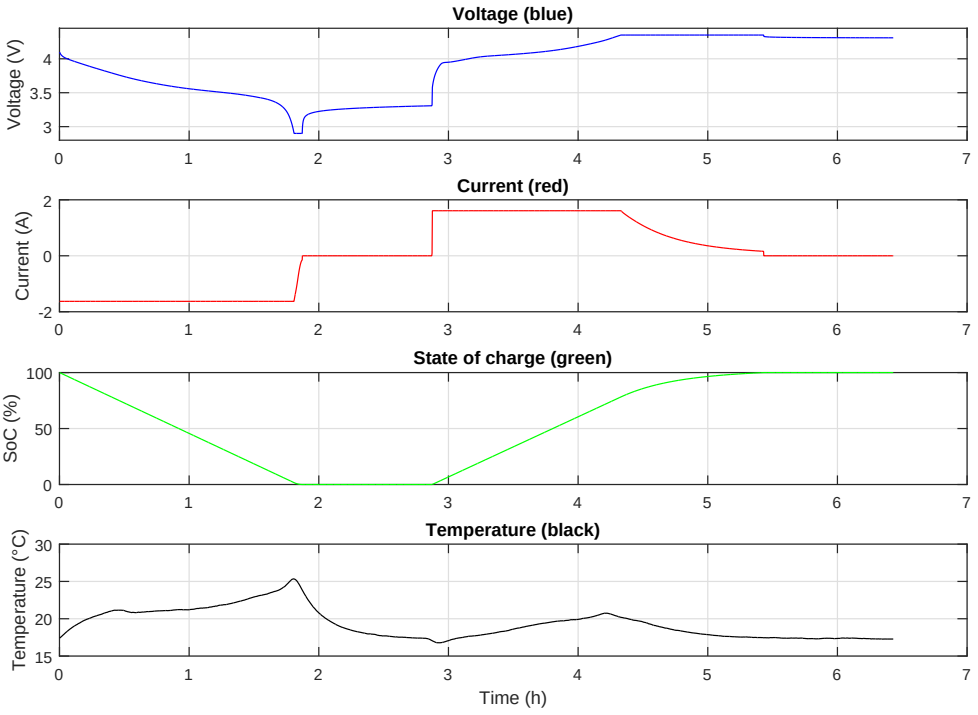

Figure 20: Training cycle results.

Let's observe the points in which the SOC assumes the values of 100 % and 0 %. From the measurements of the current acquired during the test it is possible to extract

firstly the value of the real maximum capacity of the battery, relative to the previously introduced definitions of the state of charge limits, and then the trend of the state of charge. The SOC trend has been computed by integrating the current with the trapezoidal integration rule.

$$SOC[k+1] = SOC[k] + \frac{100 \cdot T_{sampling}}{2 \cdot Q_{max}} \left(I_{batt}[k+1] + I_{batt}[k] \right) \tag{8}$$

The value of the maximum capacity extracted from the last cycle of the initial training is of 2993 mA h. This value is used in the following pulsed current test as battery maximum capacity, instead of the value of the nominal capacity provided by the manufacturer.

As previously introduced, the samples of the measured temperature are used in the tests only for safety. For this reason, the trend of the temperature is not considered in the following part of the work.

3.3 Pulsed current test

After the battery initial training, the pulsed current test can be actually realized. The pulsed current test is used for the battery characterization and it aims at the successive extraction of the parameters of the electrical equivalent circuit of the battery. The process of parameter extraction will be presented in section 4. In this section is instead presented the procedure with whom the test is realized in laboratory, again using Python language.

The experimental setup is the same used for the battery initial training. In the pulsed current test, it is important to observe the behaviour of the battery voltage during the transients and for this reason, a sampling frequency of 10 Hz is used for the data acquisition. As seen for the battery initial training, also the pulsed current test is performed at room temperature and the acquisition of the temperature during the test is used only for safety issues.

The pulsed current test is a frequently used test for the characterization of the components values of an electrical equivalent model of a battery. It has been firstly presented in [15]. From that moment, many researchers have adopted this approach, although with slightly different characteristics, to create equivalent circuit models for Li-ion batteries.

In this version, the test is composed by two different phases.

- Firstly a complete cycle of CC-CV discharge is performed with a current equal to C/2 up to an EOD voltage of 2.90 V, in order to reach a SOC of 0 %. Then an hour of pause is observed, in which the current is set to zero. After that, the battery is subjected to a complete cycle of charge, again with a current equal to C/2, up to an EOC voltage of 4.35 V in order to reach a SOC of 100 %. Then, an hour of pause is again observed in order that the transient phenomena of the

battery have expired before starting the real test. This initial phase allows the pulsed current test to start with the battery in a well known condition.

- Then the real PCT is performed. Also in this phase, a value of C/2 is used for the battery current. The entire process of discharge and charge is repeated a second time. The discharging and charging phases are however stopped for an hour every 5 % of the SOC. An hour of pause is also applied between the charge and the discharge.

In the following, a detailed description of test procedure is presented. Python scripting is again used to control the data acquisition from the instruments. Also in this case, in the initial part of the Python script that realizes the test, the test parameters are listed. The list of these parameters is shown below.

```
# Test parameters.
Q_MAX = 2.993              # Battery real capacity @(C/2, 4.35 V, 2.9 V, C/20) in Ah.
V_LIMIT_UP = 4.4           # Maximum acceptable value of the voltage.
V_LIMIT_LOW = 2.85         # Minimum acceptable value of the voltage.
I_LIMIT = 1.8              # Maximum acceptable value of the current.
TEMP_MAX = 30.0            # Maximum acceptable value of the temperature.
V_EOC = 4.35               # End of charge voltage value to set in charging.
V_EOD = 2.9                # End of discharge voltage value to set in discharging.
I_1C = Q_MAX               # 1C rate current.
I_COMPL = I_1C / 2         # Current compliance in charging and discharging.
I_EOCV = I_1C / 20         # End of constant voltage phase current.
V_RANGE = 10.0             # Voltage measurement range.
I_RANGE = 5.0              # Current measurement range.
D_SOC = 5.0                # Percentage of charge relative to each pulse.

# Pulse duration.
T_PULSE = (3600 * Q_MAX * D_SOC) / (100 * I_COMPL)

# Resting period duration.
T_REST = 3600.0
```

Let's observe that as maximum capacity `Q_max`, the value of 2993 mA h, extracted from the initial training test, is used.

The Python description of the pulsed current test procedure is immediate.

- A first `for` loop is used to repeat twenty times the cycle consisting of a discharging pulse followed by a rest period. Both the current pulse and the rest period have a precise duration. The pulse duration depends on the percentage of the SOC that you want to cover with each pulse, given the value of C/2 used as battery current. As introduced previously, the percentage of the SOC relative to each pulse is 5 % and then the pulse duration, that can be calculated as indicated in the parameter list, has a value of 6 min. For the rest period is instead chosen a value of an hour, in order to leave completely extinguish the relaxation transients.

- A second `for` loop is then used to repeat twenty times the cycle consisting of a charging pulse and a pause. As seen for the discharging phase, also in this the duration of the discharge pulse and that of the rest period are respectively 6 min and an hour.

In the figure 21, the whole test procedure is presented as a flowchart.

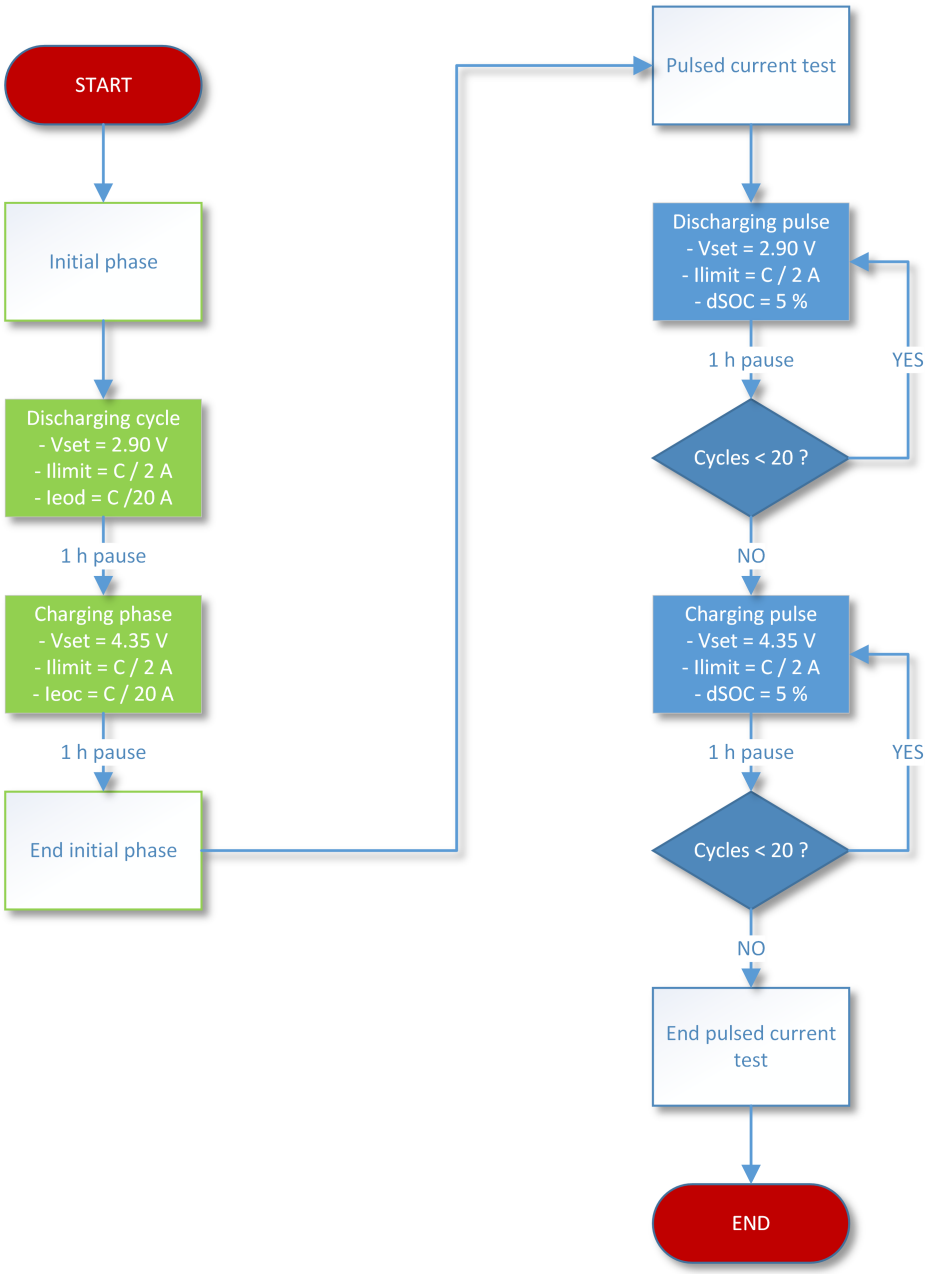

Figure 21: Pulsed current test flowchart.

The configurations of the `Keithley 2440` source-meter during the pulsed current test relative to each phase of the test are the same that are described for the initial training of the battery. Also the same control measures have been adopted to allow the progress of the test in safety.

The results of the pulsed current test are reported in the following. Firstly, the trends of the battery voltage and the battery current are shown. Their profiles are represented respectively in the figure 22 and 23.

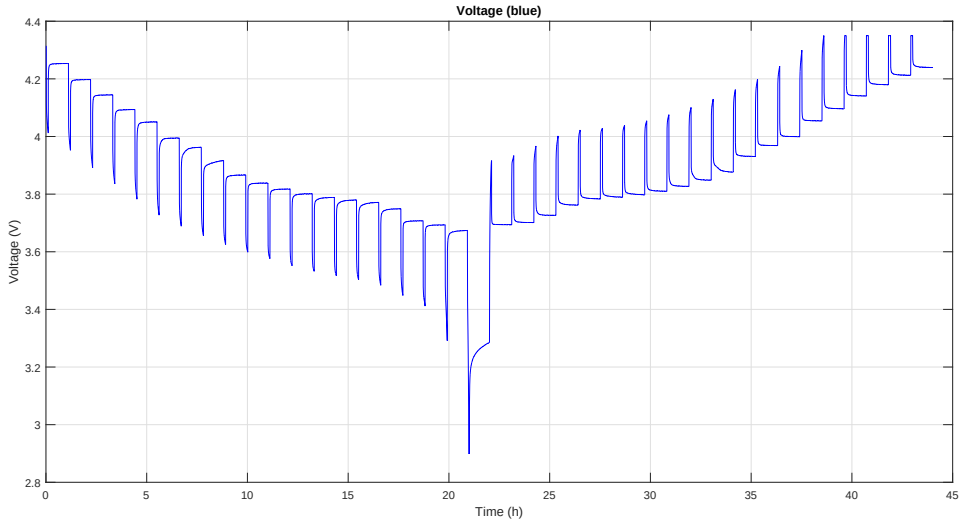

Figure 22: Pulsed current test voltage.

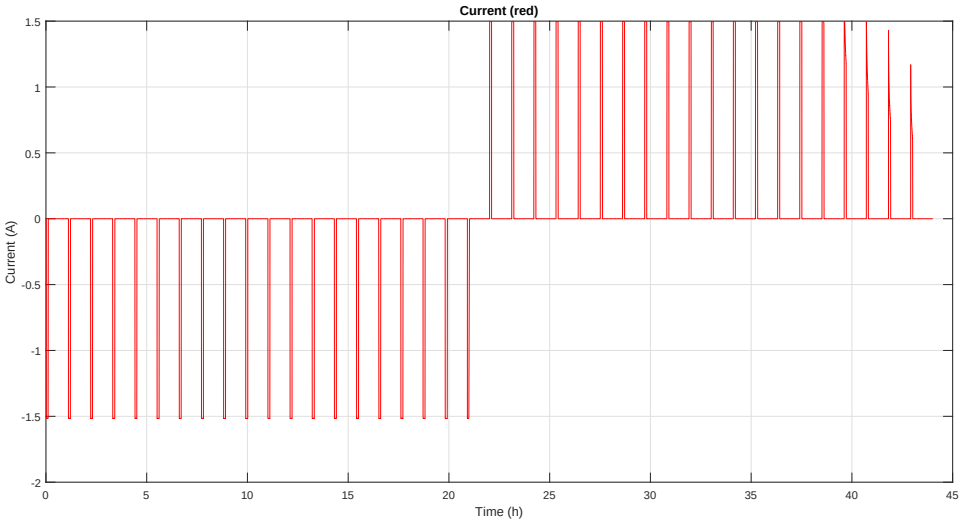

Figure 23: Pulsed current test current.

Again, as seen for the training test, from the acquired profile of the current, it is possible to extract the value of the maximum capacity of the battery actually used in the pulsed current test and then the profile of the SOC. The SOC is obtained again by integrating the current with the trapezoidal rule of integration.

The value of the maximum capacity extracted from the PCT is of about 3011 mA h. Let's annotate this value. This value, in fact, will be used as maximum capacity of the battery models implemented in the section 5.

Figure 24 finally shows the trend of the state of charge during the pulsed current test.

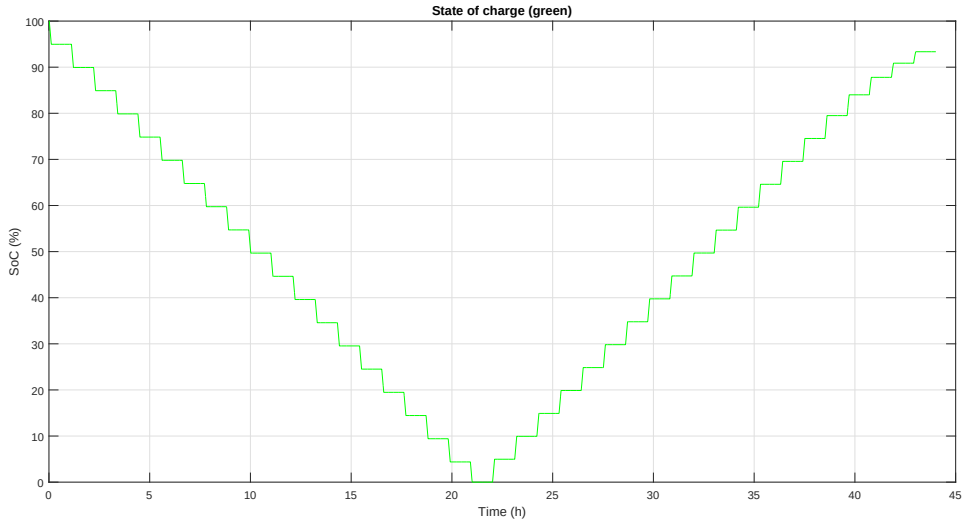

Figure 24: Pulsed current test state of charge.

4 Model parameter estimation

4.1 Parameter estimation basics

In this section, the estimation the ECM parameters of the battery is presented. The reference model is that presented in [7] and it has been introduced in section 2.

The parameters of the ECM to be estimated in this phase are $V_{open-circuit}$, R_{series}, $R_{transient-short}$, $C_{transient-short}$, $R_{transient-long}$, $C_{transient-long}$. As introduced in the section 2, the physical behaviour of the battery generally depends on state of charge, temperature and current rate. So, it is possible to write:

$$V_{open-circuit} = V_{open-circuit}(SOC, I_{battery}, T) \tag{9}$$

$$R_{series} = R_{series}(SOC, I_{battery}, T) \tag{10}$$

$$R_{transient-short} = R_{transient-short}(SOC, I_{battery}, T) \tag{11}$$

$$C_{transient-short} = C_{transient-short}(SOC, I_{battery}, T) \tag{12}$$

$$R_{transient-long} = R_{transient-long}(SOC, I_{battery}, T) \tag{13}$$

$$C_{transient-long} = C_{transient-long}(SOC, I_{battery}, T) \tag{14}$$

To investigate the behaviour of the battery in respect to the temperature and the current rate, a huge test campaign should have been done. The purpose of this work, however, is to create a battery model for simulations and not to completely characterize a specific cell. For this reason, as described in the section 3, the pulsed current test has been realized only at room temperature and using only a battery current of value C/2.

Since the pulsed current test has been carried out only at room temperature, then the ECM parameters can be identified only at room temperature. In the same way, since the pulsed current test has been realized only using a current of C/2, then the model parameters can be identified only at that current rate. Consequently, in addiction to the dependency of the ECM parameters from the battery SOC, only the dependency of the parameters from the current direction can be considered.

For this reason, two different sets of parameters are extracted in this phase. In both the sets, the parameters depends only upon the SOC. The first set is relative to the charge, while the second is relative to the discharge.

The charging parameters can be written as:

$$V_{open-circuit-charging} = V_{open-circuit-charging}(SOC) \tag{15}$$

$$R_{series-charging} = R_{series-charging}(SOC) \tag{16}$$

$$R_{transient-short-charging} = R_{transient-short-charging}(SOC) \tag{17}$$

$$C_{transient-short-charging} = C_{transient-short-charging}(SOC) \tag{18}$$

$$R_{transient-long-charging} = R_{transient-long-charging}(SOC) \tag{19}$$

$$C_{transient-long-charging} = C_{transient-long-charging}(SOC) \tag{20}$$

And equally the discharging parameters can be written as:

$$V_{open-circuit-discharging} = V_{open-circuit-discharging}(SOC) \tag{21}$$

$$R_{series-discharging} = R_{series-discharging}(SOC) \tag{22}$$

$$R_{transient-short-discharging} = R_{transient-short-discharging}(SOC) \tag{23}$$

$$C_{transient-short-discharging} = C_{transient-short-discharging}(SOC) \tag{24}$$

$$R_{transient-long-discharging} = R_{transient-long-discharging}(SOC) \tag{25}$$

$$C_{transient-long-discharging} = C_{transient-long-discharging}(SOC) \tag{26}$$

The parameters of the model are represented in this work as lookup tables. So two lookup tables, one for the discharge and the other for the charge, define the trend of each parameter of the equivalent circuit model in respect to the SOC. The size of the lookup tables depends on the resolution of the SOC used in the PCT. Since the pulsed current test has been realized with a the SOC resolution of 5 %, each lookup table relative to each parameter, both for discharge and charge, contains 21 points, from 0 % to SOC = 100 % per steps of 5 %.

4.2 Conventional estimation method

In this section, the conventional method of estimation of the battery model parameters is presented. According to this method, once the pulsed current test has been realized, it is possible to extract the values of the ECM parameters analysing the transient of the battery voltage during the pauses.

Figure 25 shows the trend of the battery voltage during a relaxation period of the pulsed current test. The shown relaxation period is that related of the 85 % of the state of charge during discharging.

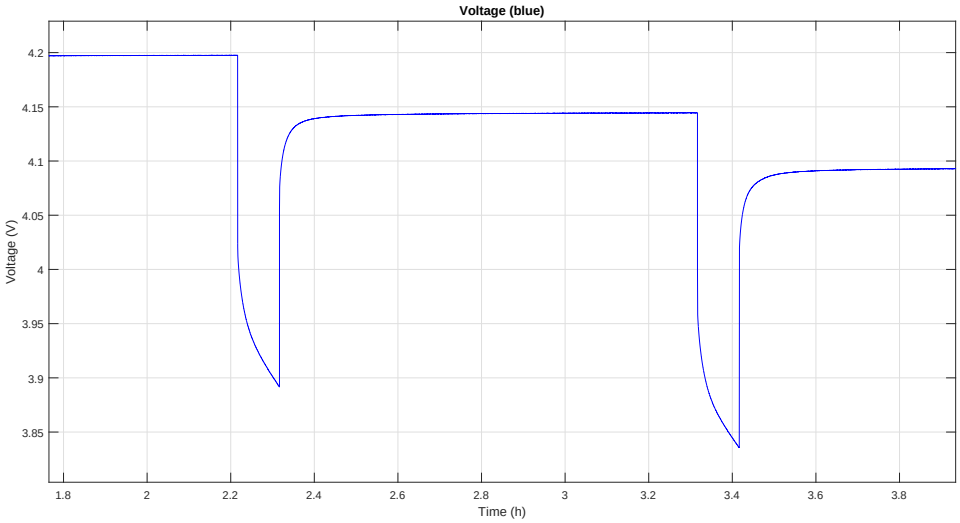

Figure 25: Relaxation period at SOC = 85 % during discharge.

During each relaxation period, the value of the current is zero. Since the current is zero, during the relaxation periods the SOC remains constant. Moreover, the SOC of the battery, during the test, changes at each pulse per steps of 5 %. For this reason, each relaxation period, both for the discharging and for the charging phase, refers to a precise and constant value of the SOC. The conventional method of parameter estimation exploits this property of the pulsed current test.

This approach is again described in [15], mentioned in the previous section 3 about the pulsed current test. The conventional estimation method is quite simple and, as mentioned above, it is based on the observation of the battery output voltage during the relation periods. Given the equivalent circuit model of the battery, the output voltage in the relaxation periods can be divided into different contributions related to each of the model parameters.

Let's consider the previous figure. The initial jump of the voltage, corresponding to the transition of the current from the nominal value to zero, can be considered in the battery model due to the series resistance. In the same way, the steady-state value of the battery voltage, reached at the end of the relaxation, can be considered as the open-circuit voltage. Then, the trend of the voltage between the initial jump and the achievement of the steady-state value, instead, can be associated to a sum of two exponential terms, represented in the equivalent model by the two R-C groups. Finally, the jump of the voltage due to the transition of the current from zero to the nominal value can be associate again to the series resistance. As described above, all the parameter values related to the considered relaxation transient are referred to a specific value of the state of charge, that does not change during the transient.

Figure 26 shows the various contributions of the model parameters to the battery voltage. The transient is the same shown in the figure 25 and it is referred to a SOC of 85 %.

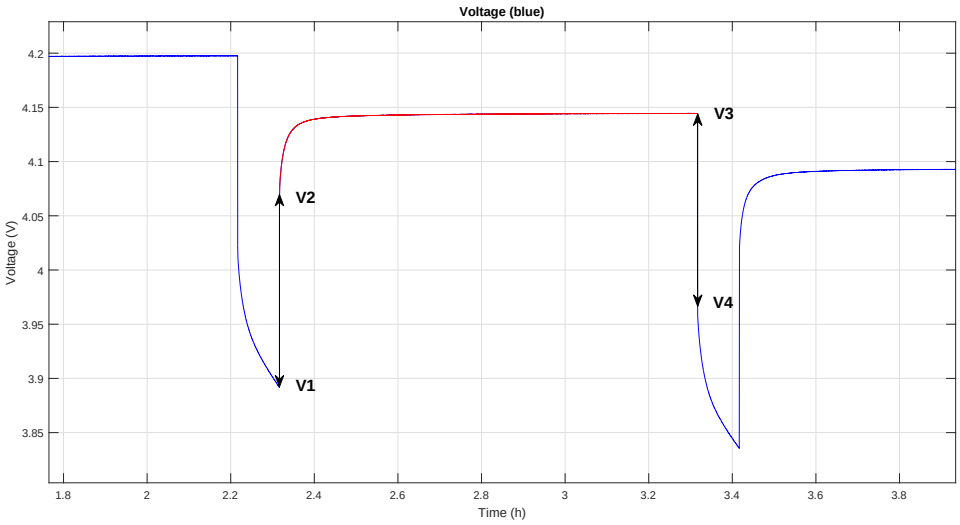

Figure 26: Different contributions to the battery voltage in relaxation.

Let's consider that the relaxation process in figure 26 corresponds to a generic state of charge $\overline{SOC}$. In the following, the algorithm that allows the estimation of the values of the battery model parameters is presented in detail.

Firstly, the value of the series resistance relative to the state of charge $\overline{SOC}$ can be extracted using the relation below.

$$\Delta V_{21} = \Delta V_{34} = R_{series}(\overline{SOC}) \cdot I_{batt} \tag{27}$$

Then, the trend of the voltage between the two steps due to the contribution of the series resistance, can be written as follow.

$$V_{23}(t) = A_{short} \exp\left\{ -\frac{t}{\tau_{short}} \right\} + A_{long} \exp\left\{ -\frac{t}{\tau_{long}} \right\} + B \tag{28}$$

In this equation, the steady-state value of the voltage $V_{23}(t)$ corresponds to the term B. So it is possible to extract the value of the OCV relative to $\overline{SOC}$ using the equation below.

$$V_{open-circuit}(\overline{SOC}) = B \tag{29}$$

It is important to observe that, if the relaxation period is long enough to permit to the battery voltage to reach the its steady-state value, then $B = V_3$. In this case the OCV can be extracted in a very simple way.

Now, let's consider the exponential terms. The terms marked with *short* are referred to the shorter time-constant, while those marked with *long* to the longer one. So, from A_{short} and τ_{short} it is possible to extract the values of $R_{transient-short}(\overline{SOC})$ and $C_{transient-short}(\overline{SOC})$ while, from A_{long} and τ_{long}, it is possible to extract the values of $R_{transient-long}(\overline{SOC})$ and $C_{transient-long}(\overline{SOC})$.

The expressions that put in relation the values of the R-C parameters relative to $\overline{SOC}$ with the values of A_{short}, τ_{short}, A_{long} and τ_{long}, again relative to $\overline{SOC}$ are reported below.

$$\tau_{short} = R_{transient-short}(\overline{SOC}) \cdot C_{transient-short}(\overline{SOC}) \tag{30}$$

$$\tau_{long} = R_{transient-long}(\overline{SOC}) \cdot C_{transient-long}(\overline{SOC}) \tag{31}$$

$$A_{short} = -R_{transient-short}(\overline{SOC}) \cdot I_{battery} \tag{32}$$

$$A_{long} = -R_{transient-long}(\overline{SOC}) \cdot I_{battery} \tag{33}$$

So in this mode, from the trend of the battery voltage during the relaxation period relative to the state of charge $\overline{SOC}$, it is possible to obtain the values of the parameters of the equivalent circuit model relative to the state of charge $\overline{SOC}$.

Repeating the estimation algorithm for each point of the state of charge, from SOC = 0 % to SOC = 100 % per steps of 5 %, firstly for the discharging phase and then for the charging, the trend of each parameter can be obtained and so the lookup tables can be populated.

To realize the parameter estimation procedure, Matlab has been used. In particular, since the estimation process is a problem of optimization, a special tool of the Matlab environment has been used, that is the Optimization Toolbox.

Figures 27, 28, 29, 30, 31 and 32 show the trends of the ECM parameter values, with respect to the SOC. In each figure, the trend of the charging parameter is shown in red, while that of the discharging parameter is shown in blue.

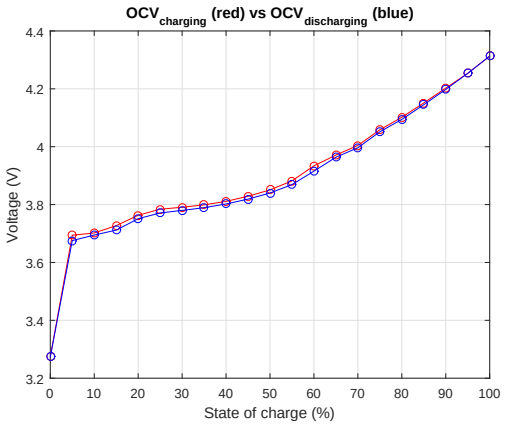

Figure 27: Open-circuit voltage.

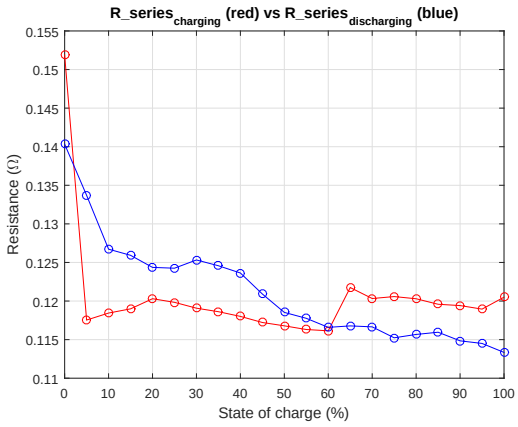

Figure 28: Series resistance.

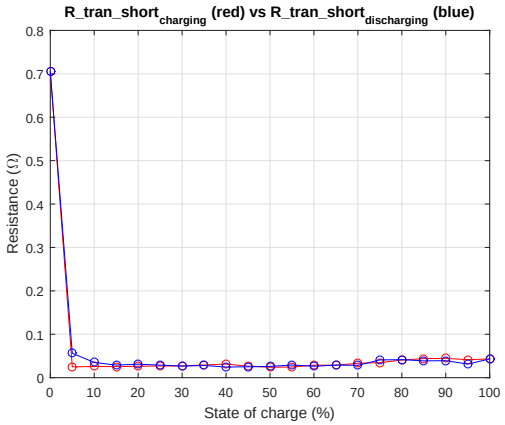

Figure 29: Short transient resistance.

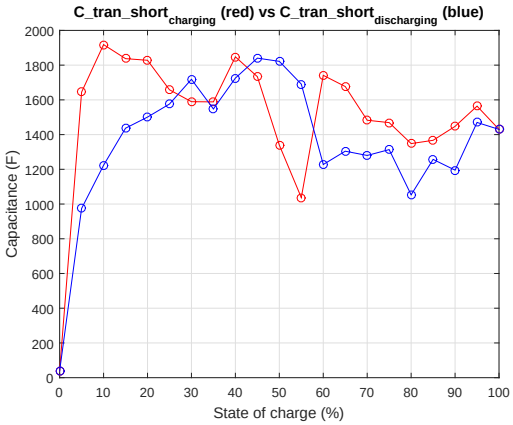

Figure 30: Short transient capacitance.

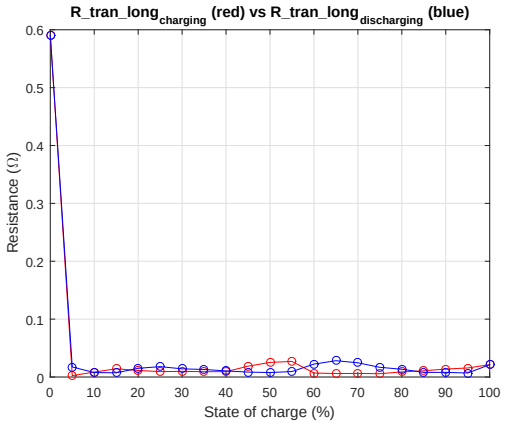

Figure 31: Long transient resistance.

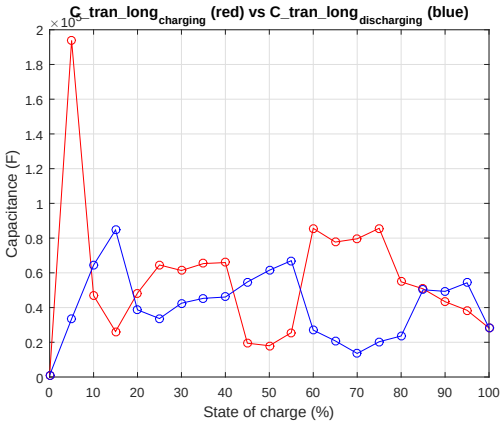

Figure 32: Long transient capacitance.

Let's introduce the Simulink model of the battery [16]. The Simulink model is introduced in this section in order to evaluate the error introduced by the conventional estimation algorithm, evaluated on the whole pulsed current test. The Simulink model is shown in the figure 33.

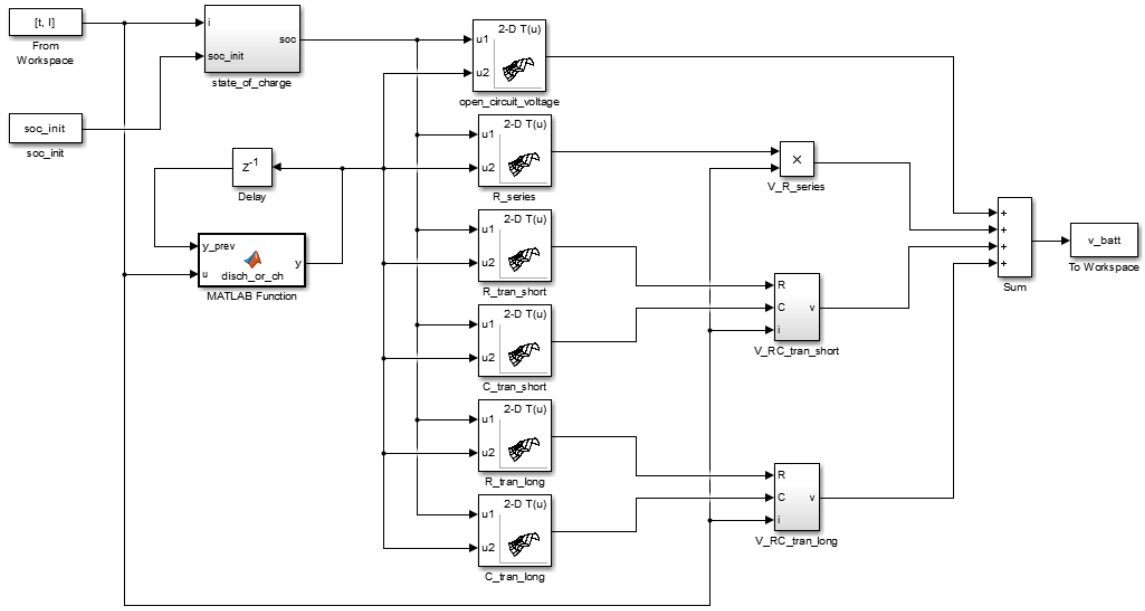

Figure 33: Simulink model.

As shown, the model presents two different input signals. The first is represented by the battery input current. This input needs to be provided to the model as a time-series, describing the trend of the current as function of time. The second input, instead, corresponds to the initial state of charge of the battery model.

As mentioned above, the introduction of the Simulink model aims at evaluating the error introduced by the conventional estimation algorithm. For this reason, the model input current corresponds to the current extracted from the pulsed current test during the battery characterization. Moreover, in the simulation of the characterization test, `soc_init` = 100 because of the battery is completely charge at the beginning of the test.

The output of the Simulink model is represented by the output voltage, that is basically the voltage at the battery terminals. Like the input current, also the output voltage is expressed as a function of the time.

Let's observe the central blocks of the model. They represent the lookup tables of the battery model parameters. From the higher towards the lower, the lookup tables relative to the open-circuit voltage, to the series resistance, to the resistance and capacitance related to the shorter time-constant and to those related to the longer time-constant are shown.

Since the model considers the dependency of the parameters both from the SOC and from the current direction, the lookup tables cannot be one-dimensional, but rather two-dimensional. The first dimension of the lookup tables describes the dependency of the model parameters from the SOC.

The SOC is computed during simulations by the subsystem present in the input

stage of the battery model. It is computed as a function of the input current and of the initial SOC. Knowing the trend of the input current in respect to the time and the cell capacity, in fact, the SOC can be computed using the Coulomb counting method.

The figure 34 shows the content of the subsystem that realizes the calculation of the state of charge.

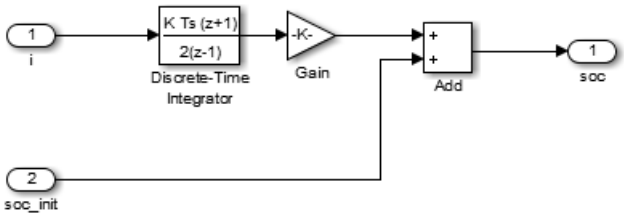

Figure 34: State of charge calculation.

The model is discrete-time. Since the acquisition of the data during the battery characterization has been realized using a sampling time of 0.1 s, also for the Simulink model sampling time the value of 0.1 s is considered. Since the model is discrete-time, the continuous time integral relative to the Coulomb counting method is approximated again with the trapezoidal rule of integration.

As introduced previously, the values of the ECM parameters are represented on 21 points, from SOC = 0 % to SOC = 100 %, per steps of 5 %. This resolution is not high enough to represent correctly the relationship between the OCV and the SOC. The OCV(SOC) curve, in fact, presents a derivative of great value in the area where the SOC is lower than 10 % and so, in this interval, little variations of the SOC imply great variation of the OCV. For this reason, a SOC resolution of 5 % during the PCT is not high enough to correctly represent the OCV(SOC) curve for SOC lower than 10 %. Consequently, an error is introduced in simulation, more pronounced when the SOC presents a value lower than 10 %. However, this error can be reduced performing an interpolation of the OCV(SOC) curve using a greater number of points. This simple operation is performed on Matlab. So, the lookup tables of the OCV, of charge and discharge, are interpolated in order to obtain 1001 points, from SOC = 0 % to SOC = 100 % per step of 0.1 %. This procedure significantly reduces the error introduced in simulation. For the other parameters, the arrays are those have been extracted in the estimation phase and so they are represented on 21 points.

Let's return to the explanation of the two-dimensional lookup tables. The second dimension of the lookup tables describes the dependency of the model parameters from the direction of the current. So, if the current has a negative value, then the parameters to consider are the discharging ones, while if the current has a positive value, then the parameters to consider are the charging ones. From this point of view, the lookup tables are substantially driven by one bit, that has the value of 1 if the current is positive and the value of 0 if it is negative.

To drive this input of the lookup tables, the Matlab function block $f(x)$ has been introduced. The purpose of this block is to control the current in input to the model and decide which set of parameters is necessary to use, between the charging and the discharging. The function controls the current sign and amplitude in order to realize a sort of comparator with hysteresis. If the current is positive and has a value higher than a certain threshold, then the model lies in a charging phase and, consequently, it is necessary to use the set of parameters relative to the charge. Else if the current is negative and its value is lower than a certain threshold, then the model lies in a discharging phase and the set of parameters to be used are that relative to the discharge. Else the current presents a value between the two thresholds. In this case, the parameters are maintained to the same value relative to the previous step.

In this mode, inside the pulsed current test, if the current presents, for example, a transition from a positive value to zero, the values of the parameters are maintained to those relative to charging phase also during the relaxation period. Moreover, this behaviour is implemented also to eliminate the error introduced by the noise on the input current. In fact, if the threshold is set to zero, a fluctuation of the current value around the zero could induce the model to consider the wrong set of parameters.

During simulations, the lookup tables of the Simulink model are interpolated using a cubic interpolation method.

Let's observe now the output stage of the Simulink battery model. The output voltage of the Simulink model is computed as sum of various contributions, the open-circuit voltage, the voltage across the series resistance and the voltages across each R-C group.

The output of the lookup table that represents the open-circuit voltage goes directly in the last block that realizes the sum of the various voltages. The output of the lookup table of the series resistance, instead, before being added to the other terms, has to be multiplied for the input current to obtain the voltage across the series resistance.

Let's consider the contributions of the R-C groups. The voltage across each R-C group is computed using an appropriate subsystem. The content of the two subsystems is the same and it is shown in the figure 35.

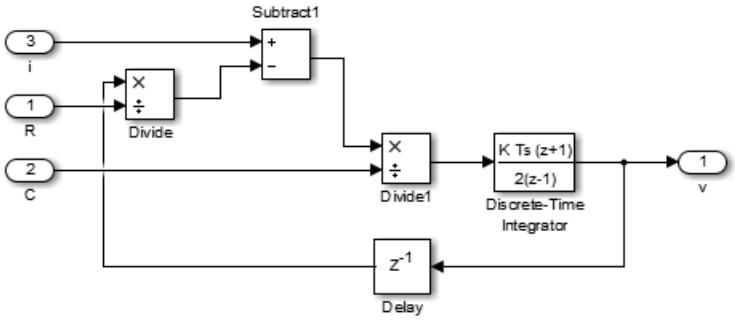

Figure 35: R-C group voltage calculation.

As seen, the subsystem has as inputs the model current and the values of resistance and capacitance of the corresponding R-C group. Using these inputs, it is able to compute the voltage at the terminals of the R-C group. The voltage can be computed as follow.

$$v(t) = v(t_0) + \frac{1}{C}\int_{t_0}^{t}\left(i - \frac{v(\tau)}{R}\right)d\tau \tag{34}$$

Since the model is discrete-time, the discretized version of this equation is really used. Once all the contributions of the various components have been computed, the battery output voltage is obtained as sum of those contributions.

It is important to observe that, since the model presents only the battery current as input, it is not possible to drive the model with a voltage. So, the Simulink model is not able to describe the behaviour of the battery when it is driven in a constant-voltage phase.

Let's pass to the description of the simulation of the Simulink model. When you launch a simulation of the model, the model output voltage is computed firstly as function of the inputs and then of the values of the parameters stored into the lookup tables.

The model inputs correspond, in this first simulation, to the electrical quantities measured during the pulsed current test. Consequently the battery voltage, that represents the Simulink model output, is computed as a function only of the estimated parameters. So, this first simulation gives information about the correctness of the parameters estimated using the conventional method.

In addition, once the Simulink model has been simulated on the pulsed current test and then the output voltage has been computed, it is possible to extract the deviation between the voltage measured during the characterization test and the simulated voltage.

The error on the voltage is defined as follow.

$$Error = \left|V_{simulation} - V_{measure}\right| \tag{35}$$

It can be expressed in percentage.

$$Error = 100\left|\frac{V_{simulation} - V_{measure}}{V_{measure}}\right| \tag{36}$$

The error introduced by the conventional estimation method is summarized in the table 2 below.

	Maximum error	Maximum percentage error	Root mean square error	Root mean square percentage error
Conventional method	447.5 mV	11.68 %	23.9 mV	0.67 %

Table 2: Error obtained after the conventional estimation method.

The root mean square error between the measured voltage and the simulated, introduced by the conventional estimation method is 23.9 mV. It is quite large and the reason is simple. As described previously, the conventional method of parameter estimation is based on fitting the battery voltage during the relaxation periods of the pulsed current test. Consequently, the conventional estimation exploits only a small part of the information potentially contained in the characterization test, just that relative to the relaxation transients.

An example of this behaviour can be observed in detail by comparing the measured voltage and the simulated. Let's consider figure 36.

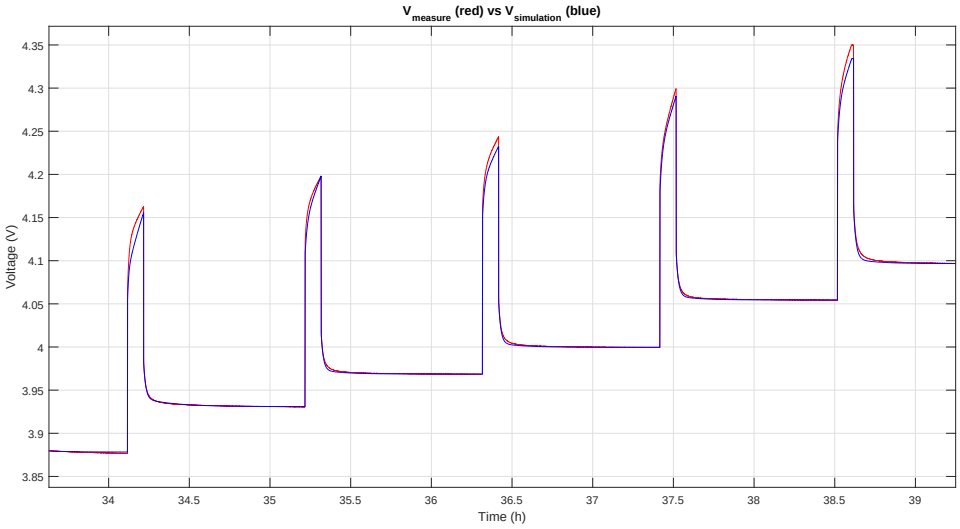

Figure 36: Measured and simulated voltage for SOC $\in$ [55 %, 80 %] in charging.

This figure shows the trend of the measured voltage and that of the simulated voltage after the conventional estimation process, evaluated in the charging phase between SOC = 55 % and SOC = 80 %.

It is possible to note that the error between the measured voltage and the simulated is small during the relaxation periods of the characterization test, while it increases when the current is not zero. This behaviour is a consequence of fitting the battery

voltage only during relaxation periods and involves a significant error between the measured voltage and the simulated.

To overcome this problem, a further step of optimization of the model parameters is introduced in the following. The optimized estimation method aims at reducing the error between the voltage measured during the characterization test and the voltage simulated with the Simulink model of the battery. Reducing the error means basically to make the behaviour of the battery model more similar to that of the real battery. To reduce the error, all the information contained in the pulsed current test are exploited in the optimized estimation method, not only that related to the relaxation periods, as seen for the conventional technique.

The optimization process is realized adding to Matlab programming, already used to realize the conventional estimation algorithm, also the Simulink battery model presented in this section, in order to obtain a mixed approach. The optimized estimation method is described in detail in the next section.

4.3 Optimized estimation method

In this section the optimization algorithm of parameter estimation is presented. This aspect is doubtless the most innovative one of this work. The optimized parameter estimation method takes inspiration from the parameter extraction algorithm presented in [17]. For the the optimization task, the Simulink model introduced in the previous section has been used.

The optimization algorithm aims at extracting the optimum values of the parameters stored into the lookup tables of the Simulink model, related respectively to R_{series}, $R_{transient-short}$, $C_{transient-short}$, $R_{transient-long}$ and $C_{transient-long}$. Two different tasks need to be performed, one to extract the discharging parameters and another to extract the charging parameters. For this purpose, the first half of the characterization test is considered to extract the discharging parameters, while the second half is considered to extract the charging parameters.

Let's observe that the $V_{open-circuit}$ is not included as parameter to be extracted. The reason is immediate. The error introduced in the relationship between OCV and SOC during the first estimation task is relatively small. So, the charging and discharging curves of the OCV are enough accurate to need no further optimization. Moreover, excluding one parameter from the optimization process, the duration of the process decreases, because the optimization tool has to account only to five parameters instead of six.

Let's describe the optimization process. The process is based on a recursive approach. At the first step of the optimization, the lookup tables related to the parameters R_{series}, $R_{transient-short}$, $C_{transient-short}$, $R_{transient-long}$ and $C_{transient-long}$ have to be initialized to determined values, while the $V_{open-circuit}$ is set to the value extracted in the previous conventional estimation technique. The inputs of the model correspond again to those measured during the pulsed current test and they remain fixed during the optimization process.

At this point, the simulation of the Simulink model of the battery can be launched and consequently, the model output voltage is computed. As introduced previously, when you launch a simulation of the Simulink model, the output voltage is computed firstly as function of the inputs and then of the values of the parameters stored into the lookup tables. Since the inputs of the model are fixed, the output voltage is a function only of the parameter values. Comparing the output voltage, simulated with the model, with the real voltage, obtained from the pulsed current test, it is possible to extract a measure of the correctness of the parameters values used in the simulation.

The optimization process aims at minimizing the least square error between the measured voltage and the simulated. Minimizing that error, in fact, means to find the optimum values of the parameters that make the behaviour of the Simulink model equal that of the real battery. So, the optimization tool, at each step of the process, automatically changes the parameter values until it finds a local minimum of the least square error between the measured voltage and the simulated.

As mentioned above, Matlab provides a specific tool for optimization problems, which is called exactly Optimization Toolbox. Between the various functions that this tool provides, the only one that has been used in this work is the function `lsqnonlin()`.

The syntax to use to call the function is reported below.

```
x = lsqnonlin(@obj_function, x0, lb, ub, opts, t_meas, v_meas, i_meas, soc_meas);
```

Now, a brief description of the function `lsqnonlin()` is presented. Its arguments are listed in the following.

- `t_meas`, `v_meas`, `i_meas` and `soc_meas` represent the arrays respectively of time, voltage, current and state of charge measured during the pulsed current test.
- `x` is the parameter matrix that the function `lsqnonlin()` has to optimize. This matrix is composed by the lookup tables relative to the ECM parameters R_{series}, $R_{transient-short}$, $C_{transient-short}$, $R_{transient-long}$ and $C_{transient-long}$. As introduced previously, each parameter is represented as an array of 21 values, in function of the state of charge, from SOC = 0 % to SOC = 100 % per steps of 5 %. Consequently, `x` is a 21x5 matrix.
- `x0` represents the initial value of the parameter matrix. The value of `x0` is used to initialize `x` at the first step of the optimization process.
- `obj_function` represents the function to minimize in order to find the optimum `x`. Inside the problems of operational research, it is commonly called objective function. Every time `obj_function` is called, it launches a simulation of the Simulink model with the actual values of the parameters matrix `x`. So, the output voltage relative to the current `x` can be computed. At this point, the difference between the measured voltage and the simulated can be computed and returned by the objective function in order that the caller function `lsqnonlin()` evaluates its root mean square error. At this moment, the difference between

the measured voltage and the simulated can be computed and returned by the objective function in order that its root mean square error could be computed by the `lsqnonlin()`.

- `opt` is a data structure that contains the options to be applied during the optimization task. In this case `opt` is configured in order that the optimization is stopped when the root mean square error between the measured and the simulated voltage reaches the value of 10^{-6}, or, when the norm of the difference between the values of the parameters relative to consecutive steps reaches again 10^{-6}.

- `ub` and `lb` are respectively the upper bound and the lower bound that the parameters can assume. Let's introduce an important concept. The optimization tool looks for just a mathematical solution to the problem that is placed, without taking into account that the solution is acceptable from a physical point of view. Consequently, the tool must be properly guided to find not only a mathematical solution, but rather to find a solution that represents the battery behaviour also from a physical point of view. For, this reason, is important to provide the right bounds to the parameters. In this work, as lower bound is used a scaled version of the parameters extracted in the conventional estimation task and as upper, an amplified version. In this way, the optimization tool is guided to a right physical solution.

As introduced above, the function `lsqnonlin()` allows to find the optimum value of `x`, corresponding to the parameter matrix to estimate, that minimizes, according to the least square method, the value of the quantity that the objective function `obj_function` returns at the end of its execution, that is $V_{measure} - V_{simulation}$. Once that the function `lsqnonlin()` has been called, at each step of the optimization task, it can change the value of the parameter matrix `x` in order minimize that error.

Figure 37 shows a flowchart that describes the optimized estimation method.

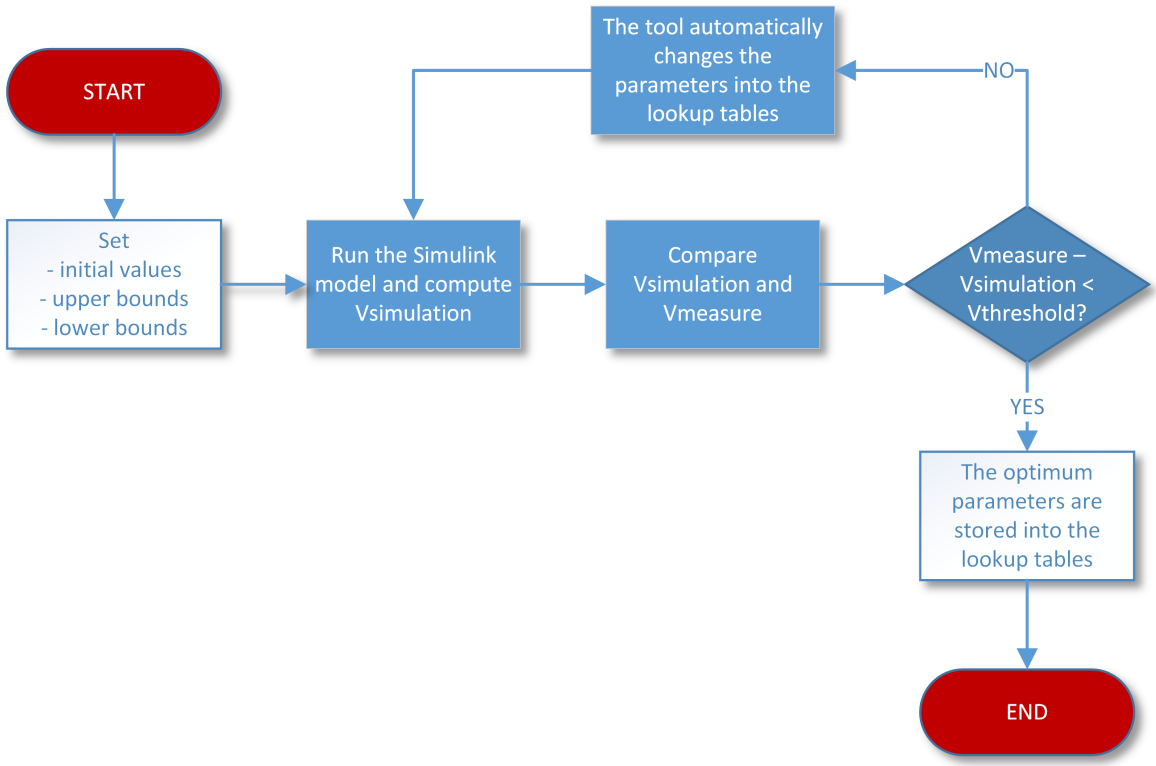

Figure 37: Flowchart representing the optimization process.

Once that the function `lsqnonlin()` has finished its execution, the optimum values of the ECM parameters have been found and they are stored into the lookup tables of the Simulink model. From there, they can be extracted and saved in order to be used for the implementation of the battery models in the analog and digital simulation environments.

4.4 Parameter extraction results

Figures 38, 39, 40, 41, 42 and 43 show the trends of the battery model parameter values computed using the optimized estimation method. Again, in each figure, the trend of the charging parameter is shown in red, while that of the discharging parameter is shown in blue.

Let's observe that the curves relative to the open-circuit voltage remain the same of those extracted using the conventional method of estimation. They are anyway reported for completeness.

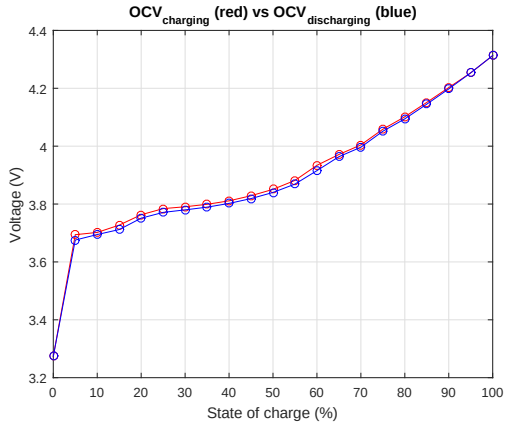

Figure 38: Optimized $V_{open-circuit}$

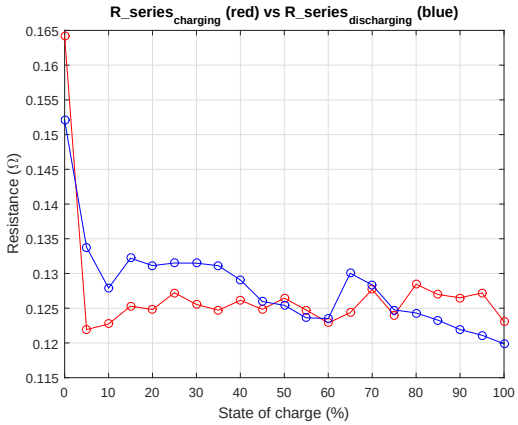

Figure 39: Optimized R_{series}.

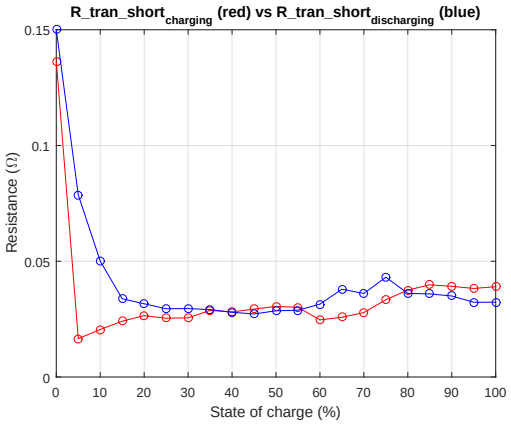

Figure 40: Optimized $R_{transient-short}$.

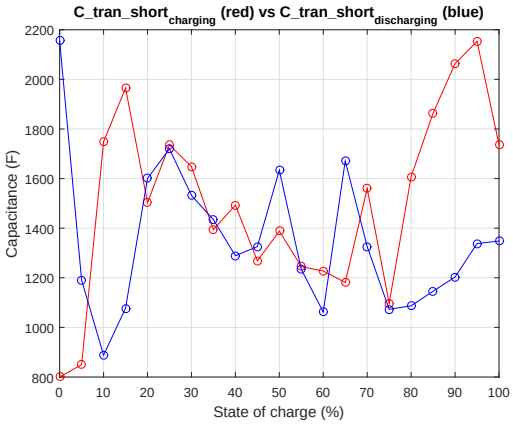

Figure 41: Optimized $C_{transient-short}$.

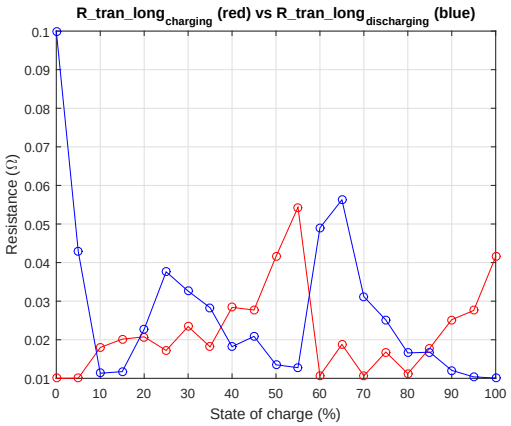

Figure 42: Optimized $R_{transient-long}$.

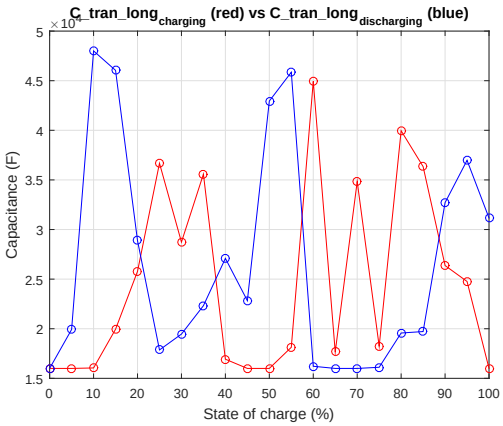

Figure 43: Optimized $C_{transient-long}$.

Similarly to the conventional estimation task, it is possible to evaluate the error between the measure voltage and the simulated. The simulated output voltage is again obtained simulating the Simulink model on all the duration of the pulsed current test, however using the parameters extracted with the optimized method.

Figure 44 shows the results of the simulation. Let's observe that the error after the optimization process seems immediately to be lower than the error evaluated at the end of the conventional estimation task.

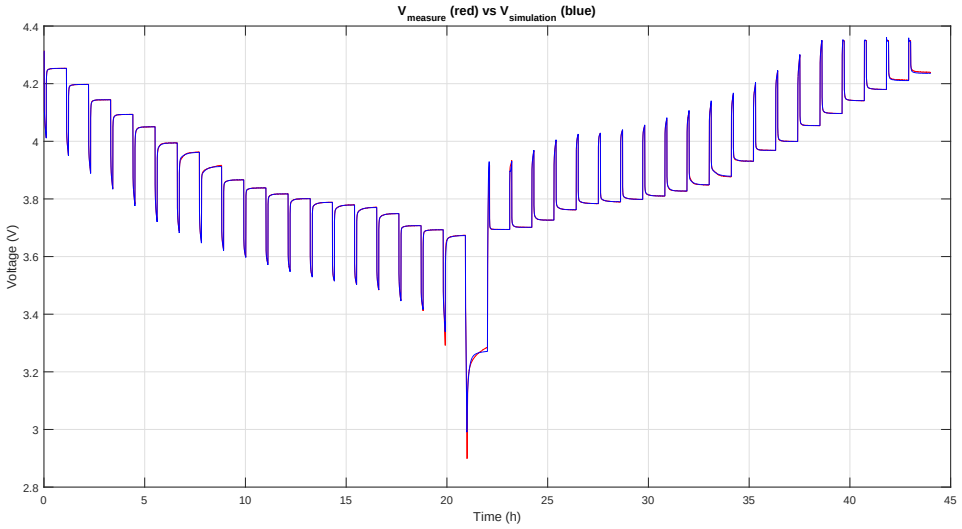

Figure 44: Simulink model simulation on PCT.

Now it is possible to extract the trend of the error $V_{measure} - V_{simulation}$. It is shown in figure 45.

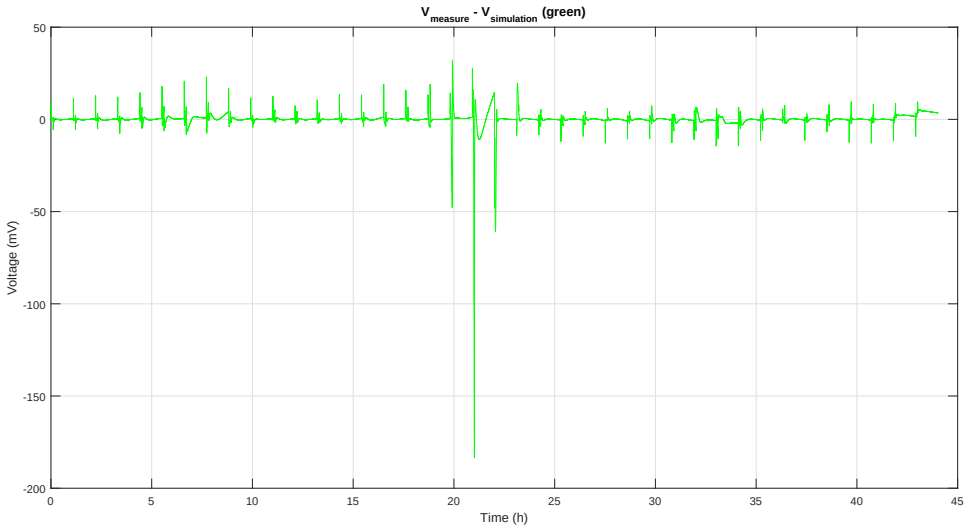

Figure 45: Simulink model simulation error on PCT.

The error is summarized in the following table 3.

	Maximum error	Maximum percentage error	Root mean square error	Root mean square percentage error
Conventional method	447.5 mV	11.68 %	23.9 mV	0.67 %
Optimized method	183.5 mV	6.33 %	4.7 mV	0.15 %

Table 3: Comparison between conventional and optimized estimation method.

As shown, after the execution of the optimization algorithm, the error $V_{measure} - V_{simulation}$ is drastically decreased in respect with that evaluated after the conventional estimation task.

Moreover, let's observe in the figures 44 and 45 that the error is mainly concentrated in the area in which the SOC is lower than 10 %.

Let's see how the error value changes considering only the area in which the SOC lies between the 10 % and the 100 %. The results obtained considering the range [10 %, 100 %] of the SOC are reported in the table 4.

Maximum error	Maximum percentage error	Root mean square error	Root mean square percentage error
23.1 mV	0.36 %	1.3 mV	0.034 %

Table 4: Error after the optimized method considering SOC $\in$ [10 %, 100 %].

This table shows that the error dramatically decreases if you consider the state of charge in the range [10 %, 100 %]. This emphasizes again the correctness of the battery model parameters extracted as result of the optimized estimation method presented in this section.

5 Model implementation

5.1 Introduction of the battery models

Once that the battery equivalent model parameters have been extracted, they can be used to implement the battery models in the different environments. In addiction to the Simulink model presented in the previous section, two other battery models have been implemented in two different simulation environments. The first is an analog model of the battery, suited for analog simulations. The tool used in this case to implement the model is Cadence Virtuoso. The second battery model is implemented using SystemVerilog HDL. This model aims at being used in digital and mixed-signals simulation. For the digital model simulations, the Cadence Incisive environment has been used.

In this section also the first simulation results are presented. Each model, in fact, after the implementation, is subjected to an initial simulation that aims at ensuring that its behaviour is correct. As described in section 4 for the Simulink battery model, also for the models presented in this section an initial verification is done using as input the current measured during the pulsed current test. Running the simulation, the value of the output voltage can be obtained. After that, the voltage obtained from the simulation of the implemented models can be compared with that obtained from the experimental test in laboratory and with that obtained from the simulation of the Simulink model. Again, the error between the measured and the simulated output voltage is extracted for each model simulation and then analysed in order to obtain information about the correct implementation of the battery model.

A detailed description of the implementations of the different battery model is provided in the following sections.

5.2 Analog battery model

5.2.1 Introduction

The first model presented in this section has been realized into the Cadence Virtuoso analog environment.

The basic idea followed for implementing the analog battery model is to describe each component of the model separately. So, an entire component library has been created, that contains the components that represent the behaviours of $V_{open-circuit}$, R_{series}, $R_{transient-short}$, $C_{transient-short}$, $R_{transient-long}$ and $C_{transient-long}$. As presented in section 4, two sets of battery model parameters have been extracted. The first set represents the charging parameters while the second represents the discharging ones. Each parameter of the sets is function only of the state of charge. So, the dependency from the state of charge and from the current direction has to be provided also to the analog model components.

Together with these, some components of the basic analog library have been used, like the capacitor that represents the cell capacity or the current-controlled current

source that decouples the part of the model that accounts for battery lifetime from the output stage.

Once the components have been created, they can be connected together and to other standard components to realize the battery model exactly as an electrical circuit.

The special components, to correctly represent the behaviour of the battery model, are described using Verilog-A [18].

Verilog-A is a modelling language, specialized for the description of analog components and circuits. It represents the continuous-time subset of Verilog-AMS. In turn Verilog-AMS is a derivative of the Verilog HDL that includes analog and mixed-signal extensions in order to define the behaviour of analog and mixed-signal systems. It extends the event-based simulator loops of Verilog/SystemVerilog/VHDL, by a continuous-time simulator, which solves the differential equations in analog-domain.

In this way, once the components $V_{open-circuit}$, R_{series}, $R_{transient-short}$, $C_{transient-short}$, $R_{transient-long}$, $C_{transient-long}$ have been described, the battery model can be mounted in a schematic view.

In the following, the implementation of each component used to realize the analog battery model is presented. Firstly, a general description of the characteristics common to all the Verilog-A components is provided. All the components present three different nodes: the positive terminal, the negative terminal, and a control input. The role of the positive and negative terminals is immediate. The current flowing across the terminals of a component, from the positive to the negative, causes a positive voltage drop between them, and vice versa. Instead, there is no current flowing through the control input. The control input, in fact, is driven by an equivalent voltage that reflects the SOC of the battery. As presented in the description of the battery ECM in section 2, the SOC is represented by the voltage on the capacitor $C_{capacity}$. This voltage ranges between 0 V and 1 V and represents exactly the SOC of the battery ECM. Let's call this voltage V_{soc}. Then, if $V_{soc} = 1$, then SOC = 100 % and the battery model is considered fully charged, else if $V_{soc} = 0$, then SOC = 0 % and it is fully discharged. The equivalent voltage V_{soc} is used as control input in order to give to the components the information about the actual state of charge.

As in the Simulink model, the trends of the components values are represented using lookup tables. Again, two different lookup tables are used to describe the behaviour that each component in function of the state of charge, the first in discharging and the second in charging. These lookup tables are addressed using the value of the control input voltage V_{soc}. So, the knowledge of the actual SOC is needed to the components to consider the right value of the component inside the correct lookup table. In turn, the correct lookup table between the charging and the discharging can be chosen by controlling the direction of the current flowing across the component terminals.

In the following, the Verilog-A description of each component is examined.

5.2.2 Component `r_series`

Let's consider firstly the description of the behaviour of the series resistance. The descriptions of the other parameters are very similar to this. Below, the initial part of the description of the component `r_series` is shown, in which there is the list of the external nodes of the component.

```
module r_series(p, n, soc);

    inout p, n;
    electrical p, n;

    input soc;
    voltage soc;

    string file_name;

    real res;
```

The nodes `p` and `n` represent respectively the positive and the negative terminals of the component. They are declared as `inout` and additionally they are declared as `electrical`. The declaration of the nodes as `electrical` means that the function `V(p, n)` and `I(p, n)` can be applied to the node. These function are specific of Verilog-A and provide respectively the value of the voltage between the nodes `p` and `n` and that of the current flowing from `p` to `n`.

The node `soc` is the control input of the component associated to the SOC of the battery model. It is an `input` and it is declared as `voltage`. The difference between a `voltage` node and an `electrical` node is that only the function `V()` can be applied to a node declared as `voltage`. Basically `V(soc)` corresponds to the voltage V_{soc} introduced earlier.

The `string filename` corresponds, instant by instant during the battery model simulation, to the path of the file containing the current lookup table to be considered between the charging and the discharging. The files specified by the path contained in `filename` are text files that describe the trend of the series resistance value in respect to the SOC, respectively for charge and for discharge. The file needs to be organized as a matrix in which the first column represents the breakpoints of the SOC, while the second represents the values of the component corresponding to the SOC breakpoints. In this way, the lookup tables relative to the series resistance can be associated to the component in order to be used during the battery model simulation.

The node `res` is declared as `real` and represents, during simulations, the current value of the series resistance, corresponding to the current state of charge.

The behavioural description of the component needs to be included inside an `analog` block. At the initial step of each simulation, the direction of the current that flows across the series resistance is checked.

```
        @(initial_step) begin
            if(I(p, n) > 0)
                file_name =
                    "/users/mrognini/battery_model_sim/estimated_params/Ro_ch.txt";
            else
```

```
            file_name =
                "/users/mrognini/battery_model_sim/estimated_params/Ro_disch.txt";
        res = $table_model(V(soc), file_name, "3");
    end
```

If the current is positive, then the battery model lies in a charging phase and then the value of the variable `filename` has to be initialized with the path of the file containing the charging lookup table. Else if the current is negative or zero, then the battery model lies in a discharging phase and so `filename` has to be initialized with the path of the file of the discharging lookup table.

Let's introduce the function `$tablemodel()`, on which the design of the components is based. This function presents three different arguments.

- `V(soc)` is the voltage of the input node `soc`, that represents the SOC.
- `filename` contains the path of the file corresponding to the lookup table related to the current phase of the model.
- The third argument allows to set the lookup table interpolation and extrapolation method to be used for the component. In this case `"3"` means cubic interpolation and linear out-of-range extrapolation.

Specifying these arguments, the value of the series resistance corresponding to `V(soc)` can be extracted from the lookup table related to the current phase of the model and assigned to the variable `res`.

Now, the central part of the behavioural description of the component is presented.

```
    // If a negative transition of the current occurs, switch to discharge state.
    @(cross(I(p, n) + 0.000001, - 1))
        file_name =
            "/users/mrognini/battery_model_sim/estimated_params/Ro_disch.txt";
    // If a positive transition of the current occurs, switch to charge state.
    @(cross(I(p, n) - 0.000001, + 1))
        file_name =
            "/users/mrognini/battery_model_sim/estimated_params/Ro_ch.txt";

    res = $table_model(V(soc), file_name, "3");
    V(p, n) <+ res * I(p, n);
```

At each step of the simulation process, the value of the current direction is checked. If the value of the current passes through 10 µA upward, then the component recognizes a charging phase and consequently, the path of the charging lookup table is assigned to the variable `filename`. If instead the value of the current passes through -10 µA downward, then the component recognizes a discharging phase and consequently, the path of the discharging lookup table is assigned to the variable `filename`. If the value of the current lies between the two thresholds, then the value of the variable `filename` is maintained.

Once the value of the variable `filename` has been assigned, the value of the series resistance `res` can be computed using again the function `$table_model()`. Finally,

once `res` has been updated, the voltage between the nodes `p` and `n` of the component is computed using the Ohm's law.

In the next sections, the descriptions of the other components of the battery model are presented. Since these descriptions are very similar to that of the series resistance, only the differences between them and the description of the series resistance are reported.

5.2.3 Component `open_circuit_voltage`

In this section is presented the description in Verilog-A of the open-circuit generator. The whole description is inserted below.

```
module ocv_soc_lut(p, n, soc);

    inout p, n;
    electrical p, n;

    input soc;
    voltage soc;

    string file_name;

    analog begin

        @(initial_step) begin
            if(I(p, n) > 0)
                file_name =
                    "/users/mrognini/battery_model_sim/estimated_params/ocv_ch.txt";
            else
                file_name =
                    "/users/mrognini/battery_model_sim/estimated_params/ocv_disch.txt";
        end

        // If a negative transition of the current occurs, switch to discharge state.
        @(cross(I(p, n) + 0.000001, - 1))
            file_name =
                "/users/mrognini/battery_model_sim/estimated_params/ocv_disch.txt";
        // If a positive transition of the current occurs, switch to charge state.
        @(cross(I(p, n) - 0.000001, + 1))
            file_name =
                "/users/mrognini/battery_model_sim/estimated_params/ocv_ch.txt";

        V(p, n) <+ $table_model(V(soc), file_name, "3");

    end

endmodule
```

As shown, the structure of the component description is the same of that presented earlier for the series resistance. In this case, the lookup tables provide directly the value of the OCV corresponding to the value `V(soc)`. So, once that the variable `filename` has been updated with the path of the file that contains the lookup table of the current phase of the battery model, the return value of the function `$table_model()` can be assigned directly to the voltage across the nodes `p` and `n`.

5.2.4 Component `r_tran_short`

The component `r_tran_short` corresponds to the resistor of the R-C group that presents the shorter time-constant. Its description in Verilog-A is very similar to that relative to the series resistance.

```
module r_tran_short(p, n, soc);

    inout p, n;
    electrical p, n;

    input soc;
    voltage soc;

    string file_name;

    real res;

    analog begin

        @(initial_step) begin
            if(I(p, n) > 0)
                file_name =
                    "/users/mrognini/battery_model_sim/estimated_params/Rs_ch.txt";
            else
                file_name =
                    "/users/mrognini/battery_model_sim/estimated_params/Rs_disch.txt";
            res = $table_model(V(soc), file_name, "3");
        end

        // If a negative transition of the current occurs, switch to discharge state.
        @(cross(I(p, n) + 0.000001, - 1))
            file_name =
                "/users/mrognini/battery_model_sim/estimated_params/Rs_disch.txt";
        // If a positive transition of the current occurs, switch to charge state.
        @(cross(I(p, n) - 0.000001, + 1))
            file_name =
                "/users/mrognini/battery_model_sim/estimated_params/Rs_ch.txt";

        res = $table_model(V(soc), file_name, "3");
        V(p, n) <+ res * I(p, n);

    end

endmodule
```

The only differences between the description of the `r_tran_short` and that of the `r_series` are basically represented by the paths to consider in order to load the values of the components from the lookups tables.

5.2.5 Component `c_tran_short`

The component `c_tran_short` corresponds to the capacitor of the R-C group that presents the shorter time-constant. Let's show the description of this component.

```
module c_tran_short(p, n, soc);

    inout p, n;
    electrical p, n;
```

```
    input soc;
    voltage soc;

    string file_name;

    real cap;

    analog begin

        @(initial_step) begin
            if(I(p, n) > 0)
                file_name =
                    "/users/mrognini/battery_model_sim/estimated_params/Cs_ch.txt";
            else
                file_name =
                    "/users/mrognini/battery_model_sim/estimated_params/Cs_disch.txt";
            cap = $table_model(V(soc), file_name, "3");
        end

        // If a negative transition of the current occurs, switch to discharge state.
        @(cross(I(p, n) + 0.000001, - 1))
            file_name =
                "/users/mrognini/battery_model_sim/estimated_params/Cs_disch.txt";
        // If a positive transition of the current occurs, switch to charge state.
        @(cross(I(p, n) - 0.000001, + 1))
            file_name =
                "/users/mrognini/battery_model_sim/estimated_params/Cs_ch.txt";

        cap = $table_model(V(soc), file_name, "3");
        I(p, n) <+ cap * ddt(V(p, n));

    end

endmodule
```

Let's observe that the initial part of the description is equal to that of the other components presented so far. In this case, from the lookup tables, the value of the capacitance is loaded. So, the value of the current that flows across the component terminals can be computed with the constitutive equation of a capacitor. The operator `ddt()` computes the derivative of the quantity passed to it as argument.

5.2.6 Component `r_tran_long`

The component `r_tran_long` corresponds to the resistor of the R-C group that presents the longer time-constant. Its description is reported for completeness.

```
module r_tran_long(p, n, soc);

    inout p, n;
    electrical p, n;

    input soc;
    voltage soc;

    string file_name;

    real res;

    analog begin
```

```
        @(initial_step) begin
            if(I(p, n) > 0)
                file_name =
                    "/users/mrognini/battery_model_sim/estimated_params/Rl_ch.txt";
            else
                file_name =
                    "/users/mrognini/battery_model_sim/estimated_params/Rl_disch.txt";
            res = $table_model(V(soc), file_name, "3");
        end

        // If a negative transition of the current occurs, switch to discharge state.
        @(cross(I(p, n) + 0.000001, - 1))
            file_name =
                "/users/mrognini/battery_model_sim/estimated_params/Rl_disch.txt";
        // If a positive transition of the current occurs, switch to charge state.
        @(cross(I(p, n) - 0.000001, + 1))
            file_name =
                "/users/mrognini/battery_model_sim/estimated_params/Rl_ch.txt";

        res = $table_model(V(soc), file_name, "3");
        V(p, n) <+ res * I(p, n);

    end

endmodule
```

5.2.7 Component `c_tran_long`

The last component to describe is `c_tran_long`. It represents the capacitor relative to the R-C group corresponding the longer time-constant. Its Verilog-A description is reported below for completeness.

```
module c_tran_long(p, n, soc);

    inout p, n;
    electrical p, n;

    input soc;
    voltage soc;

    string file_name;

    real cap;

    analog begin

        @(initial_step) begin
            if(I(p, n) > 0)
                file_name =
                    "/users/mrognini/battery_model_sim/estimated_params/Cl_ch.txt";
            else
                file_name =
                    "/users/mrognini/battery_model_sim/estimated_params/Cl_disch.txt";
            cap = $table_model(V(soc), file_name, "3");
        end

        // If a negative transition of the current occurs, switch to discharge state.
        @(cross(I(p, n) + 0.000001, - 1))
            file_name =
                "/users/mrognini/battery_model_sim/estimated_params/Cl_disch.txt";
        // If a positive transition of the current occurs, switch to charge state.
        @(cross(I(p, n) - 0.000001, + 1))
```

```
            file_name =
                "/users/mrognini/battery_model_sim/estimated_params/Cl_ch.txt";

        cap = $table_model(V(soc), file_name, "3");
        I(p, n) <+ cap * ddt(V(p, n));

    end

endmodule
```

5.2.8 Analog model schematic view

Once the components of the ECM have been described, it is possible to extract their schematic symbols from the Verilog-A descriptions. After that, these components are ready to be placed into the schematic circuit that represents the battery model.

The analog equivalent circuit model of the battery is shown in figure 46.

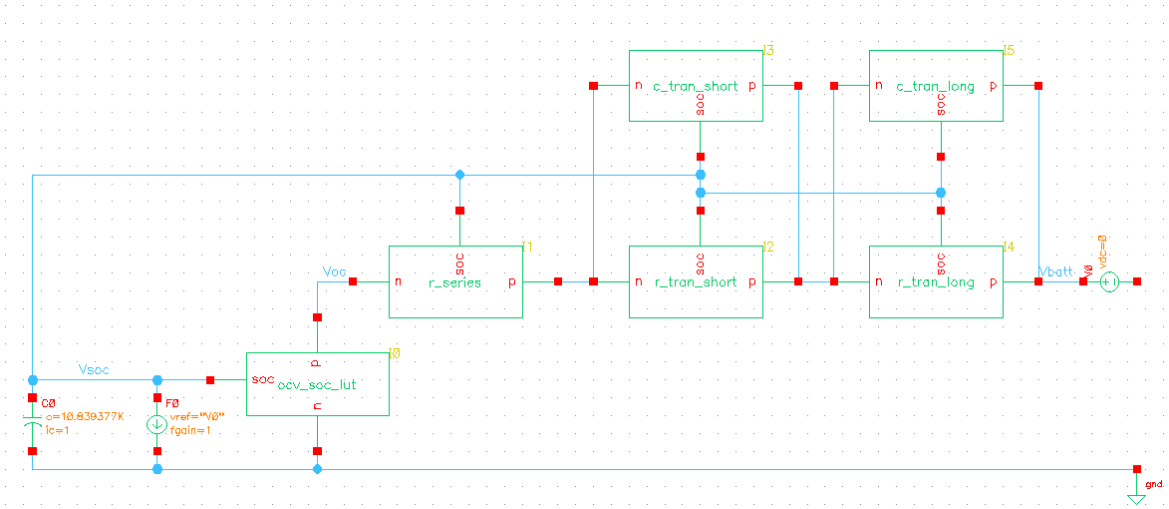

Figure 46: Analog battery model.

As shown, in addition to the components described earlier, other standard components are placed into the equivalent circuit model.

The capacitor `C0` represents the battery maximum capacity and has a capacitance of 10 839.377 F. That capacitance of `C0` has been obtained as the maximum capacity extracted during the parameter estimation process from the pulsed current test, of about 3011 mA h, divided by 1 V. In this mode, the voltage on this capacitor, that is V_{soc}, ranges between 0 V and 1 V and represents the battery model SOC.

The other standard component inserted corresponds to the current-controlled current source that charges and discharges the capacitor `C0`. The control current needs to be specified as the current that flows in a voltage generator. For this reason, the dummy voltage source `V0` has been inserted. So, the current-controlled generator is controlled by the current that flows across the dummy voltage source `V0`, that coincides to the input current.

Let's observe that all the control inputs `soc` related to each component are connected together and to the positive terminal of the capacitor `C0`.

So, every component is connected to the others as expected. Let's observe that the aspect of the ECM presented is practically equal to that relative to the reference battery model presented in the section 2.

It is important to observe that, since the analog battery model is circuital, it is possible to drive it both with a current source and with a voltage source. So, the implemented model can be driven to realize both a constant-current phase and a constant-voltage phase.

Let's introduce another aspect of the implementation of this model. During the battery characterization phase, the thresholds related to the 0 % and to the 100 % of the SOC have been established. In particular, the SOC is considered at 100 % when, during charging, the current reaches the value of C/20, after being charged with a CC-CV approach, at a current rate of C/2 up to a voltage of 4.35 V. Vice versa, the SOC is considered at 0 % when, during discharging, the current reaches the value of C/20, after being discharged with a CC-CV approach, at a current rate of C/2 up to a voltage of 2.90 V.

The analog model implementation makes possible to exceed the values of the SOC of 100 % during charging and of 0 % during discharging. In fact, if for example the battery model is charged using the CC-CV method, up to a voltage of 4.35 V, until the current reaches the value of zero in the CV phase, then the charge accumulated inside the cell is more than the charge used as maximum capacity, and so the state of charge correctly exceeds 100 %.

In reality, this behaviour is acceptable only for little excesses over 100 % and under 0 %. If the SOC far overcomes the limit values, the behaviour of the battery model is not guaranteed.

In addition the trends of the model parameters are unknown outside the range [0 %, 100 %] of the SOC. For this reason, as introduced previously, a linear out-of-range extrapolation algorithm has been implemented for computing the values of the equivalent parameters when the SOC exceeds the range [0 %, 100 %].

5.2.9 Analog model simulation

In this section, a first simulation of the analog battery model is presented. As described for the model implemented in the Matlab/Simulink environment, also for this the first simulation is performed using as input of the model the current extracted from the pulsed current test. In this mode, the simulated output voltage of the pulsed current test is computed and then it can be compared with the measured voltage, in order to extract the error $V_{measure} - V_{simulation}$ and verify the correctness of the analog model behaviour.

For this reason, a test bench has to be set up. The simple test bench is shown in the figure 47.

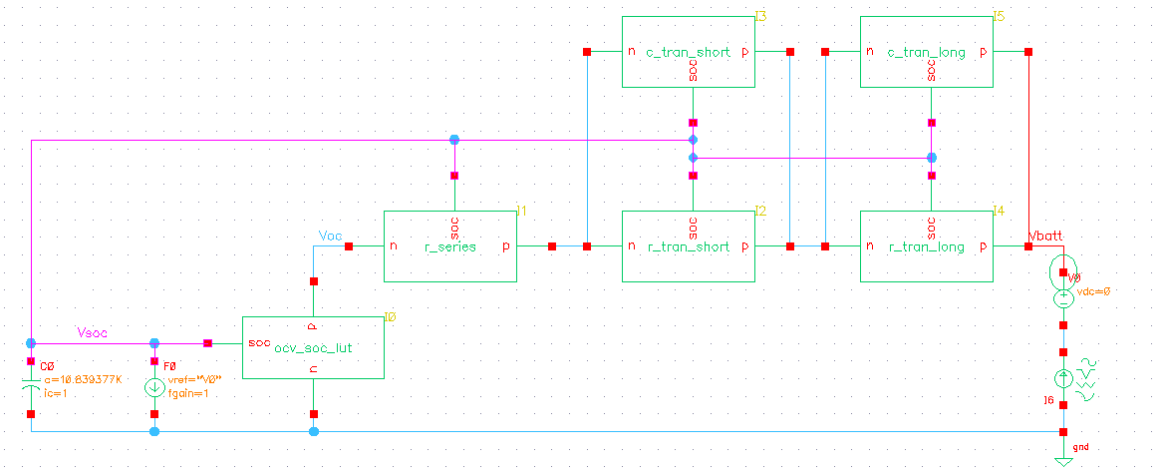

Figure 47: Analog battery model test-bench.

In the test bench, the battery model is driven by a current source. For this reason, the component `isource` of the analog standard library has been used. The component `isource` is a current source that can be configured in order to generate a specific current profile contained in an input file. The input file is of text type and it has to be organized as a time-series, in which the trend of the current provided by the current source is described in respect to the time. So, in this mode, the current generator can create the profile of the current realized during the pulsed current test in order to correctly drive the battery model.

The simulation is realized using Cadence Spectre Circuit Simulator. The result of simulation is reported in the figure 48.

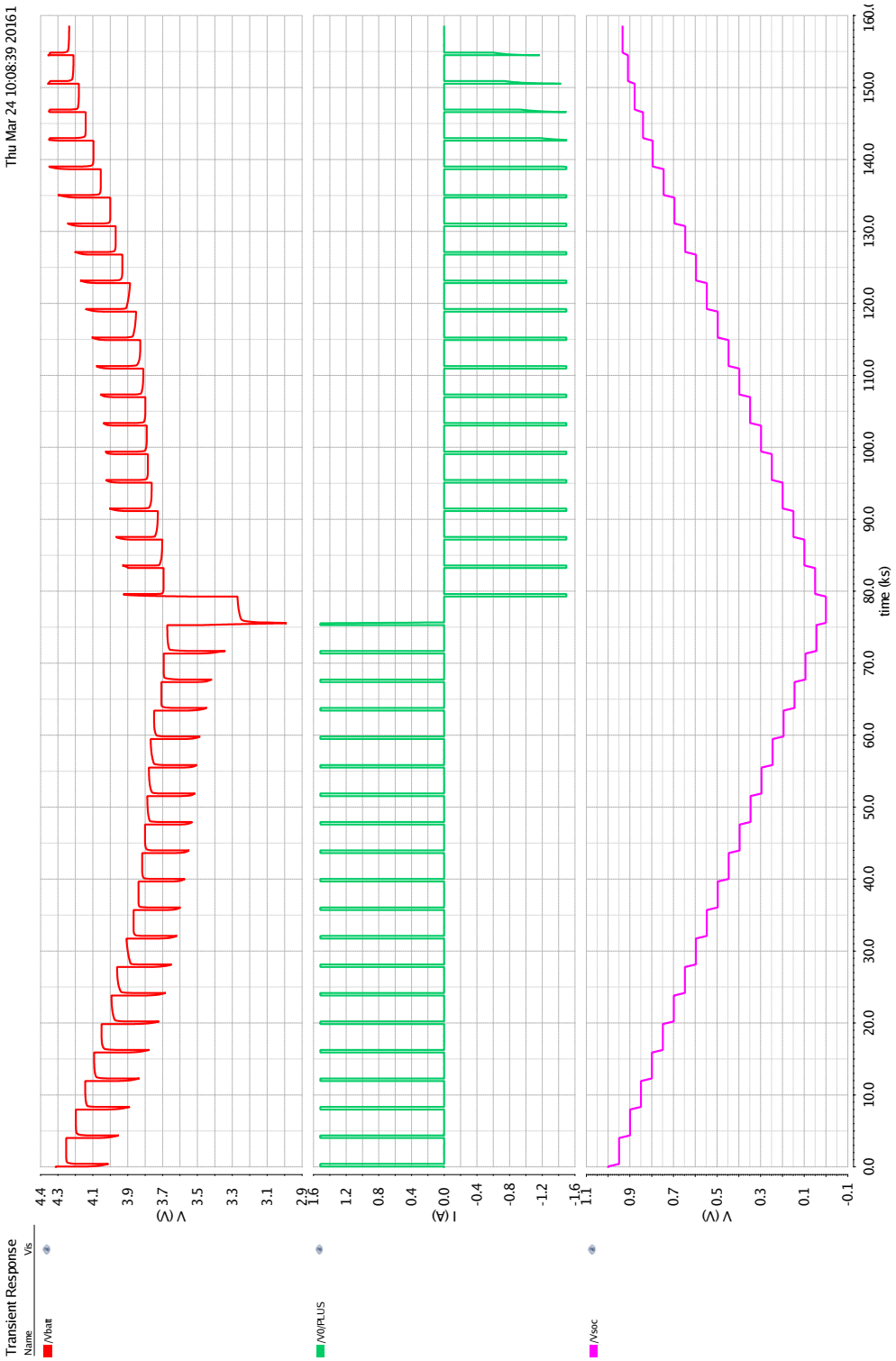

Figure 48: Analog model simulation on PCT in Cadence.

Figure 48 shows the trends respectively of the simulated voltage, current and state of charge.

Let's observe that a correspondence exists between the colors of the signal traces shown in the figure above and the colors of the nodes of the circuit represented in figure 47. It is important to observe also that the profile of the battery current is inverted in the sign in respect to that of the Simulink model. In the Cadence Virtuoso environment, in fact, the current is considered positive when it is absorbed by a generator and consequently the shown profile is that relative to the discharging current and not that relative to the charging as seen for the Simulink model.

At this point, in order to evaluate the error introduced by the analog battery model, the results of the simulation are exported from Cadence and then imported into Matlab.

Figures 49 and 50 show respectively the trend of the simulated output voltage compared with that of the measured output voltage and the trend of the error $V_{measure} - V_{simulation}$. The results are plotted in Matlab.

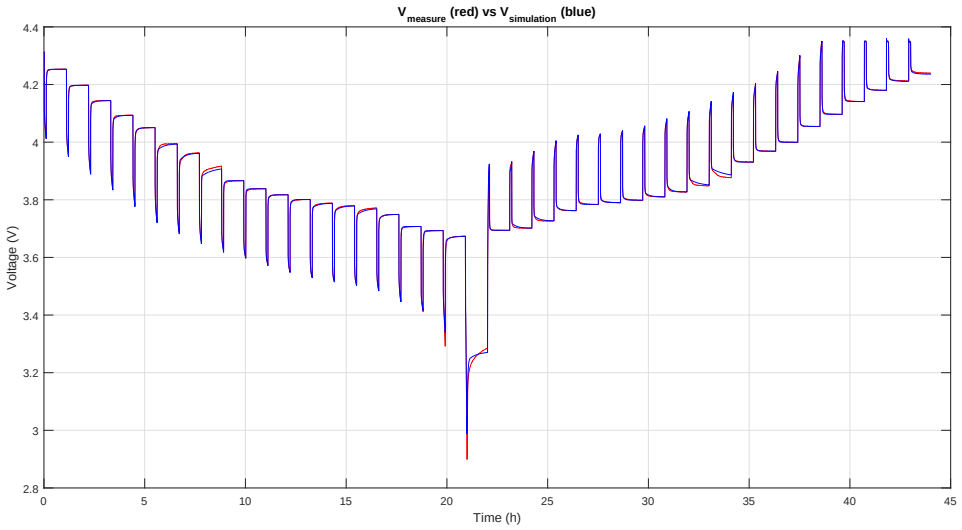

Figure 49: Analog model $V_{measure}$ vs $V_{simulation}$ on PCT.

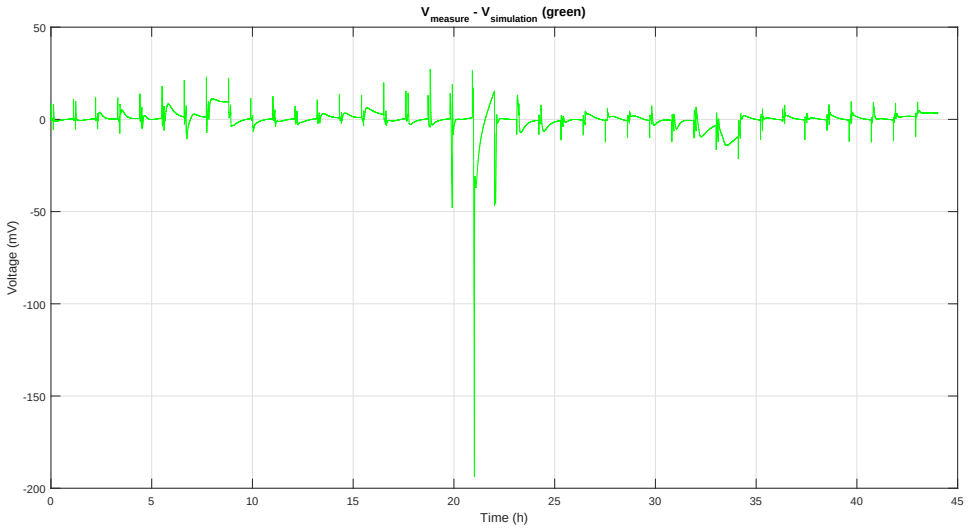

Figure 50: Analog model $V_{measure} - V_{simulation}$ on PCT.

Once the simulation results have been imported in Matlab, as shown for the Simulink model, the error between the measured and the simulated voltage can be extracted.

Table 5 summarizes the error $V_{measure} - V_{simulation}$ relative to the simulation of the analog battery model on the pulsed current test.

	Maximum error	Maximum percentage error	Root mean square error	Root mean square percentage error
Simulink model	183.5 mV	6.33 %	4.7 mV	0.15 %
Analog model	193.7 mV	6.68 %	5.7 mV	0.18 %

Table 5: Simulink model vs Analog model on PCT.

As it is possible to see in the table, the value of the error of the analog battery model is comparable with those introduced by the Simulink model and so the analog model can be considered correctly implemented.

5.3 Digital/mixed-signal battery model

5.3.1 Introduction

In this section, the implementation of the second battery model is presented. The battery model described in this section aims at being used into digital and mixed-signal simulations and so it presents a discrete-time behaviour. In general, a linear

time-invariant (LTI) discrete-time dynamical system can be described with its state-space representation.

$$\begin{cases} x[k+1] = Ax[k] + Bu[k] \\ y[k] = Cx[k] + Du[k] \end{cases}$$

Let's describe the various terms.

- `x[k]` is the array that represents the state of the system at the current time step.
- `u[k]` is the array of the system inputs at the current time step.
- `y[k]` is the array of the system output at the current step. It can be computed as function of the present state `x[k]` and of the input `u[k]`.
- `x[k+1]` is the array of the system state at the next step. Like the system output, it depends on the current state and on the current input.
- `A`, `B`, `C` and `D` are the matrices that that bind the various parameters to each other. These matrices describe the rule of the system state update and of the output assignment.

The digital battery model presented in this section is based on these notions. In the battery model, the state of the system is defined by the voltage on the capacitor $C_{transient-short}$, by the voltage on the capacitor $C_{transient-long}$ and by the state of charge. Each of the variables that compose the system state is called state variable.

The battery model is implemented using SystemVerilog HDL. In this mode, it can be easily ported into digital and mixed-signal simulation environments. The core of the model is represented by the function that describe the rule of system state update and output assignment. This function is called `update_battery_model()` and it is described using C language.

To connect the C function to the SystemVerilog battery model, the SystemVerilog Direct Programming Interface has been used. The Direct Programming Interface is a functionality provided by SystemVerilog that allows you to connect the SystemVerilog world with a foreign language, mainly C [19]. The DPI-C has several advantages. It allows to reuse already existent code and to write code in classic C language. The DPI-C can be used through a C library providing specific data types and functions to handle data coming from the SystemVerilog world. It is important to note that the data-type mapping is not one-to-one, so it is necessary to pay attention in case of passing variables that not fit between the C types.

Figure 51 shows the data-type mapping between SystemVerilog and C language.

SystemVerilog	*C (input)*	*C (output)*
byte	char	char*
shortint	short int	short int*
int	int	int*
longint	long long int	long int*
shortreal	float	float*
real	double	double*
string	const char*	char**
string [N]	const char**	char**
bit	svBit or unsigned char	svBit* or unsigned char*
logic, reg	svLogic or unsigned char	svLogic* or unsigned char*
bit[N:0]	const svBitVecVal*	svBitVecVal*
reg[N:0] logic[N:0]	const svLogicVecVal*	svLogicVecVal*
unsized array[]	const svOpenArrayHandle	svOpenArrayHandle
chandle	const void*	void*

Figure 51: Data-type mapping between SystemVerilog and C language [19].

The SystemVerilog module of the battery is clocked by a signal called `sim_clock`. In correspondence with the rising edge of this signal, from inside the module, the function `update_battery_model()` is called. Once called, the function `update_battery_model()` updates the state of the model and computes its output. Essentially, the SystemVerilog external module behaves like a wrapper for the C function. How the battery parameters are exchanged between the function and the wrapper is described in detail in the next sections.

As introduced in section 2, for the simulation of the digital/mixed-signal battery model, Cadence Incisive Enterprise Simulator is used. This simulator does not support the `shortreal` type, that corresponds to `float` in C language, but it only supports the `real`. For this reason, the variables inside the function `update_battery_model()` need to be declared as `double`, in order to match with the `real` data type of the SystemVerilog world.

In the following, the implementation of the function `update_battery_model()` and of the SystemVerilog wrapper module are presented.

5.3.2 Description of the function `update_battery_model()`

As introduced above, the function `update_battery_model()` describes the rule of state update and output assignment of the digital battery model.

The C program is divided into three different files:

- `update_battery_model.c` contains only the function `update_battery_model()`.
- `battery_model_data.h` is an header file that contains the declarations of the

lookup tables related to the parameters and other constant parameters like the cell capacity.

- `battery_model_data.c` contains the definition of the parameters declared in the file `battery_model_data.h`.

As introduced above, the state variables inside this function are represented by the state of charge, the voltage across the capacitor relative to the shorter time-constant, and that across the capacitor relative to the longer time-constant.

The state of the model is updated at each call to the function. For this reason, the state variables have to be declared as `static` inside the function. In this mode, their values can be passed through consecutive calls.

```
// Model state variables.
static double state_of_charge;
static double v_rc_tran_short;
static double v_rc_tran_long;
```

The arguments of the function are the variables `v_batt*`, `i_batt`, `soc_init` and `t_update`.

The variable `v_batt*` corresponds to the value of the voltage at the battery terminals and it represents basically to the output of the function `update_battery_model()`. It is passed to the function as pointer in order that the function can modify its value in the outer SystemVerilog module that is the caller of the function.

The other arguments represent the inputs of the function `update_battery_model()`. The battery input current is represented by the variable `i_batt`. The variable `soc_init` corresponds to the initial SOC assigned to the battery model at the first call to the function. Finally, `t_update` corresponds to the time interval that passes between two consecutive state updates. It is basically the sampling time of the model.

The lookup tables related to the battery model parameters are represented on 1001 points, both for the charge and for the discharge. The reason is due to the implementation of the algorithm of lookup table interpolation. In the other implemented model, in fact, the interpolation of the parameters is cubic. The cubic interpolation, however, cannot be easily implemented using C language. So, inside the function, a linear interpolation is implemented. Consequently, in order to preserve the accuracy in the interpolation algorithm, the number of points of the lookup tables has been increased.

The interpolation of the lookup tables is performed in Matlab. The new lookup tables are computed using a cubic interpolation. In this mode, the accuracy obtained with the interpolation algorithm is almost the same of that of the other models described in the previous sections.

In the following, the description of how the model state is updated is presented. Let's consider that the battery model state at the step `[k]` is defined by the state variables:

- `state_of_charge[k]`,

- `v_rc_tran_short[k]`,
- `v_rc_tran_long[k]`.

So, the battery model parameters present well-defined values, corresponding to the current state of charge `state_of_charge[k]` and to the current phase in which the model lies, between charge and discharge. The parameter values at the step `[k]` are:

- `open_circuit_voltage[k]`,
- `r_series[k]`,
- `r_tran_short[k]`,
- `c_tran_short[k]`,
- `r_tran_long[k]`,
- `c_tran_long[k]`.

Let's consider also that the input arguments of the function `update_battery_model()` present certain values defined by:

- `i_batt`,
- `soc_init`,
- `t_update`.

Firstly, using the input arguments, the function computes the new value of the state variables:

$$state\_of\_charge[k+1] = state\_of\_charge[k] + 100\left(\frac{i\_batt \cdot t\_update}{Q\_max}\right) \quad (37)$$

$$v\_rc\_tran\_short[k+1] = \\ = v\_rc\_tran\_short[k] + \frac{t\_update}{c\_tran\_short[k]}\left(i\_batt - \frac{v\_rc\_tran\_short[k]}{r\_tran\_short[k]}\right) \quad (38)$$

$$v\_rc\_tran\_long[k+1] = \\ = v\_rc\_tran\_long[k] + \frac{t\_update}{c\_tran\_long[k]}\left(i\_batt - \frac{v\_rc\_tran\_long[k]}{r\_tran\_long[k]}\right) \quad (39)$$

Let's observe that an appropriate algorithm needs to be implemented to initialize the values of the state variables at the initial step. In particular:

- `state_of_charge[0]` = `soc_init`,
- `v_rc_tran_short[0]` = 0,
- `v_rc_tran_long[0]` = 0.

Also the values of the model parameters used at the first step need to be initialized. Their values are assigned at the initial step, after the initialization of the SOC, and are corresponding to `soc_init` and to the sign of the current `i_batt`.

At this point, the output voltage `v_batt` can be computed as the sum of the different contributions related to the OCV, to the voltage drop across the series resistance and to those across the two R-C groups.

$$v\_batt = open\_circuit\_voltage[k] + \\ + r\_series[k] \cdot i\_batt + v\_rc\_tran\_short[k+1] + v\_rc\_tran\_long[k+1] \quad (40)$$

Let's observe that an hypothesis has been done both for computing the new model state and the model output voltage. The hypothesis is that of using the Forward Euler integration method within the state update function. Using that integration method means that the values of the model parameters used for updating of the model state at the step `[k+1]` are considered constant and corresponding to those computed at the end of the step `[k]`. In the same mode also the value of `i_batt` is considered constant and represents the current flowing into the battery in the interval between the end of the step `[k]` and the beginning of the step `[k+1]`.

Since the model state related to the step `[k+1]` is computed using the values of the parameters related to the step `[k]`, the values of the parameters at the step `[k]` need to be memorized in order to be used at the step `[k+1]`. For this reason, inside the C code, it is necessary to declare the variables corresponding to the current parameters as `static`, as seen for the state variables.

```
// Model current parameters.
static double open_circuit_voltage;
static double r_series;
static double r_tran_short;
static double c_tran_short;
static double r_tran_long;
static double c_tran_long;
```

At this point, the values of the model parameters have to be updated. The new parameter values are computed as function firstly of the sign of the current `i_batt`. If `i_batt` is positive, then the model lies in a charging phase and so the charging lookup tables have to be used. If instead `i_batt` is negative, then the model lies in a discharging phase and so the discharging lookup tables have to be considered. Once the sign of the input current has been controlled, the new values of the parameters can be computed by addressing the lookup tables with the value `state_of_charge[k+1]`. Finally, at the end of its execution, the function returns `state_of_charge[k+1]`.

The behaviour of the function `update_battery_model()` is shown using a diagram in figure 52.

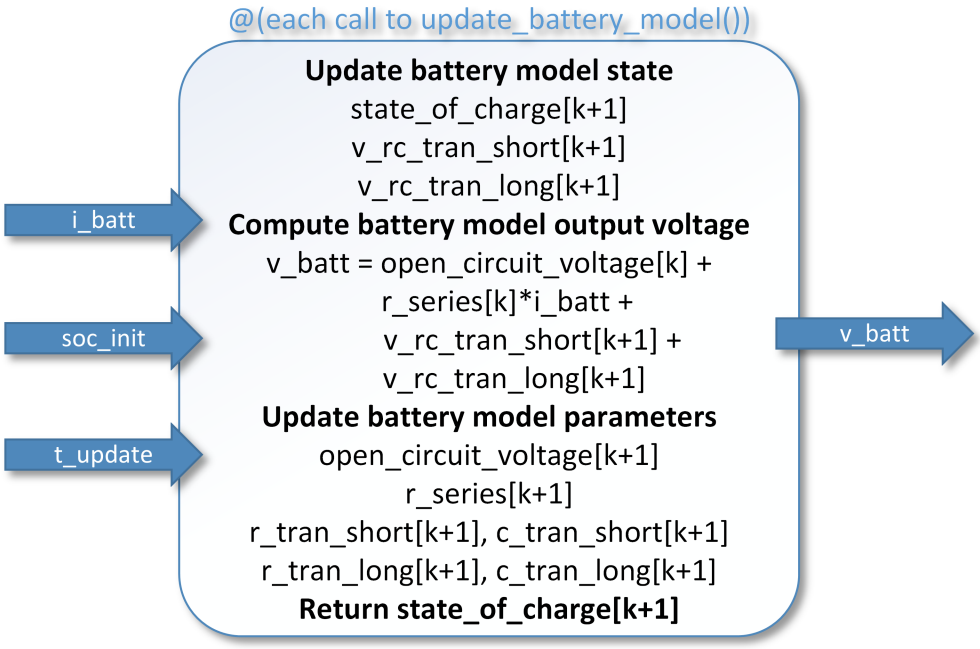

Figure 52: `update_battery_model()`.

The digital battery model, as described so far, can be driven only by the current `i_batt`. So, it is not able to represent the behaviour of the battery when this is driven by a voltage. For this reason, it can be useful to introduce in the function `update_battery_model()` the possibility of emulating the characteristics of the battery charger. Introducing the charger characteristics within the battery model basically aims at describing the behaviour of the battery model when it is driven by a voltage in a constant-voltage phase.

This behaviour already exists in the analog battery model presented in the previous section. In that case, since the analog model is circuital, there is no difference between driving it with a current or with a voltage.

In this case, the battery model is realized as a C function and so it is not really an electrical circuit. Consequently this behaviour has to be implemented as an additional feature of the model.

When the battery model is driven by a current, then the output of the model is the battery voltage. This represents the behaviour of the model already implemented. When, instead, the battery is driven by a voltage, then the output of the model is the battery current. This behaviour has to be implemented.

To consider both the two behaviours, the model has to be upgraded with a current output and with a voltage input. Consequently, other two arguments have to be included in the function `update_battery_model`.

The arguments of the function are now the variables `v_batt*`, `i_batt*`, `i_limit`, `v_set`, `soc_init` and `t_update`.

```
// Update the battery model state every t_update seconds.
double update_battery_model(double *i_batt,
                            double *v_batt,
                            const double i_limit,
                            const double v_set,
                            const double soc_init,
                            const double t_update) {
```

The variables `soc_init` and `t_update` have the same function introduced earlier and they are inputs for the function `update_battery_model()`.

With the variables `v_set` and `i_limit`, you can drive the battery model with a constant voltage of value `v_set`, providing a limit on the maximum current `i_limit`. They represent inputs for the function, like `soc_init` and `t_update`.

The variables `v_batt*` and `i_batt*` represent now the output voltage and input current of the battery model. They are outputs for the function. Again they are passed to the function as pointers in order that the function can modify their values in the outer SystemVerilog module.

A representation of the complete function `update_battery_model` is shown below in figure 53.

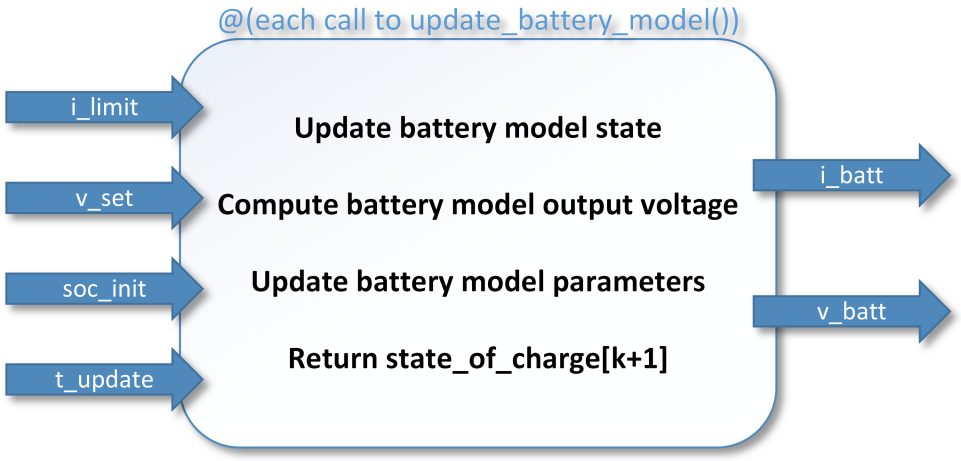

Figure 53: Complete `update_battery_model()`.

Let's consider again that the battery model state at the step `[k]` is defined by the state variables:

- `state_of_charge[k]`,
- `v_rc_tran_short[k]`,
- `v_rc_tran_long[k]`.

Inside the function, local variables called `i_in` and `v_out` are declared and used instead respectively of `i_batt*` and `v_batt*`. Obviously at the end of the function their value will be assigned to the output variables.

Firstly, the variable `i_in` is initialized at the value of the input `i_limit`. Then, the function tries to impose the current `i_in` = `i_limit` to the battery model, as if it is driven in a constant-current phase. So, the new state of the model, related to the step `[k+1]`, can be computed using the same equations presented earlier.

$$state\_of\_charge[k+1] = state\_of\_charge[k] + 100\left(\frac{i\_in \cdot t\_update}{Q\_max}\right) \tag{41}$$

$$\begin{aligned} &v\_rc\_tran\_short[k+1] = \\ &\quad = v\_rc\_tran\_short[k] + \frac{t\_update}{c\_tran\_short[k]}\left(i\_in - \frac{v\_rc\_tran\_short[k]}{r\_tran\_short[k]}\right) \end{aligned} \tag{42}$$

$$\begin{aligned} &v\_rc\_tran\_long[k+1] = \\ &\quad = v\_rc\_tran\_long[k] + \frac{t\_update}{c\_tran\_long[k]}\left(i\_in - \frac{v\_rc\_tran\_long[k]}{r\_tran\_long[k]}\right) \end{aligned} \tag{43}$$

How the update of the model state is described in the C code is shown below.

```
// Update the value of the state.
state_of_charge = state_of_charge + (i_in * 100 * t_update) / q_max;
v_rc_tran_short = v_rc_tran_short +
    ((i_in - (v_rc_tran_short / r_tran_short)) * t_update) / c_tran_short;
v_rc_tran_long = v_rc_tran_long +
    ((i_in - (v_rc_tran_long / r_tran_long)) * t_update) / c_tran_long;
```

Now, it is possible to compute the value of `v_out` obtained imposing `i_in` = `i_limit`. The equation used is the same presented in the previous part of this section.

$$\begin{aligned} &v\_out = open\_circuit\_voltage[k] + \\ &\quad + r\_series[k] \cdot i\_in + v\_rc\_tran\_short[k+1] + v\_rc\_tran\_long[k+1] \end{aligned} \tag{44}$$

Let's see the C description of this passage.

```
// Compute the model output voltage.
v_out = open_circuit_voltage + r_series * i_in + v_rc_tran_short + v_rc_tran_long;
```

Once that the output voltage `v_out` has been computed, its value has to be controlled. The model is in fact driven not only by the current `i_limit`, but also by the voltage `v_set`. Consequently the battery output voltage cannot exceed the maximum limit represented just by `v_set`. For this reason, at this point, a control has to be done.

If `v_out` is lower then `v_set`, then the battery model is effectively working in a constant-current phase, in which the value of the current `i_in` is imposed to `i_limit`.

In this case, the model next state has been correctly updated in the previous part of the function `update_battery_model()`.

Otherwise, the output voltage `v_out` is higher than the threshold `v_set`. In this case the maximum voltage limit has been exceeded. So, not the current limit `i_limit` has to be applied to the battery model, but rather the voltage limit `v_set`. Consequently, the battery model is driven in a constant-voltage phase and the output voltage `v_out` is set to `v_set` in this case.

The model state computed in the previous part imposing `i_in` = `i_limit` has not been correctly updated if the model works in a constant-voltage phase. The input current `i_in`, in fact, if the model works in a constant-voltage phase, presents a lower value than `i_limit`. This value has to be computed is this phase.

So, in this case `v_out` = `v_set`, and the model input current `i_in` needs to be computed. The value of `i_in` can be computed as follow:

$$i\_in = \\ = \frac{v\_out - open\_circuit\_voltage[k] - v\_rc\_tran\_short[k] - v\_rc\_tran\_long[k]}{r\_series[k]} \tag{45}$$

Let's observe that the state variables used for computing `i_in` are those related to the previous step `[k]` and not those just computed in the first part of the function. So, inside the function, it is necessary to memorize their values before the first state update, related to the constant-current phase, in order to use them if the model is driven in a constant-voltage phase.

Let's see how the current `i_in` is computed in the C function when the battery model is driven in a constant-voltage phase.

```
// Compute the current value in the constant voltage phase.
i_in = (v_out - v_rc_tran_short - v_rc_tran_long - open_circuit_voltage) /
    r_series;
```

At this point, both that the model has been driven in a CC phase and in a CV phase, the values of the input current `i_in` and of the output voltage `v_out` have been correctly computed.

With the real value of `i_in` just computed, the battery model state can be updated to its real value, relative to the constant-voltage phase. The equations to be used are the same seen previously. Let's observe that also the SOC is now updated at its real value. So, it is possible to compute the new values of the parameters, related to `state_of_charge[k+1]` just computed.

As described previously for the analog battery model, also for this, the state of charge can exceed the range [0 %, 100 %]. For this reason, an out-of-range extrapolation algorithm needs to be implemented inside the function `update_battery_model()` for computing the values of the model parameters when the SOC exceeds the range [0 %, 100 %]. From the characterization of the battery, no informations are extracted about the parameters trends outside this range. The design choice adopted in this model

is to maintain the parameters value equal to those related to SOC = 100 % is the SOC is higher than 100 %, and to maintain them equal to those related to SOC = 0 % is the SOC is lower than 0 %. This method of extrapolation is usually called clamp extrapolation.

At this point, once also the parameter values have been updated using the rule described above, the values of the variables `i_in` and `v_out`, declared local inside the function, are assigned to the outputs of the function, respectively `*i_batt` and `*v_batt`.

```
// Update the values of current and voltage.
*i_batt = i_in;
*v_batt = v_out;
```

Finally, the function returns `state_of_charge[k+1]` and then its execution ends.

```
// Return the actual state of charge.
return state_of_charge;
```

The algorithm performed by the C function `update_battery_model()` can be easily described as a flowchart. The flowchart that summarizes the behaviour of the function is shown in figure 54.

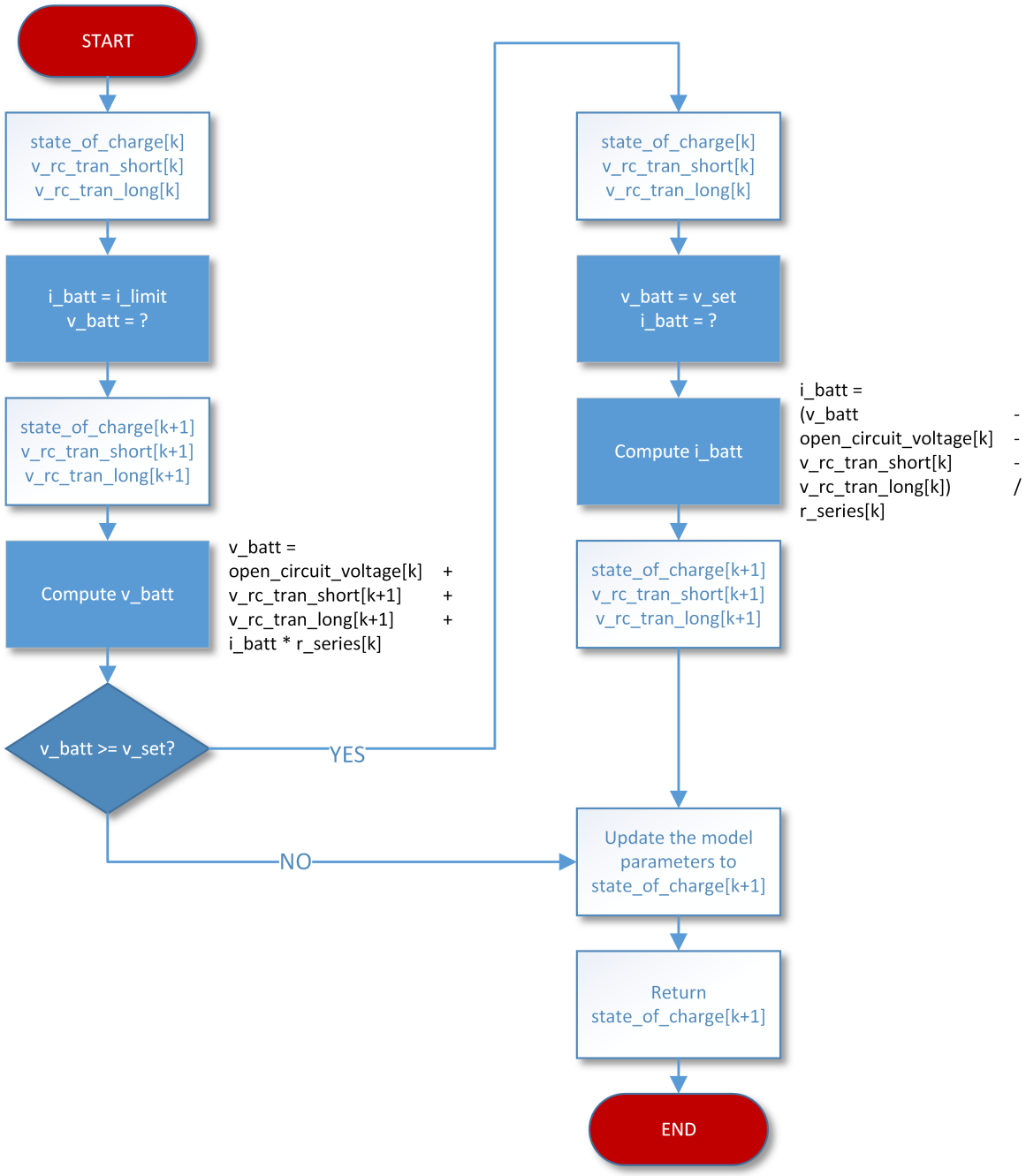

Figure 54: `update_battery_model()` flowchart.

5.3.3 Description of the SystemVerilog wrapper module

Let's move to the description of the SystemVerilog battery module. The module presents different input and output signals.

- `i_limit` is an input signal that defines the current limit with which the battery

module is driven.

- `v_set` is an input signal. It defines the voltage to set at the battery module terminals.
- `sim_clock` is an input signal. It is a digital signal that functions as a clock for the module. The SystemVerilog module is in fact a digital block and so it has to be driven by a clock signal.
- `i_batt` is an output signal that represents, instant by instant during simulations, the battery module real charging current. The current is equal to `i_limit` only if the battery module works in a constant-current phase.
- `v_batt` is an output signal that represents, instant by instant during simulations, the battery module output voltage. Let's observe that the voltage is equal to `v_set` only if the battery module works in a constant-voltage phase.
- `state_of_charge` is an output signal that corresponds to the SOC of the battery module.

The module, as described earlier, behaves basically as a wrapper for the C function `update_battery_model()`. The function can be imported into the SystemVerilog world using the special construct `import`. The way to import the C function into the SystemVerilog battery module is represented below.

```
import "DPI" function real update_battery_model(output real battery_current,
                                                output real battery_voltage,
                                                input real  current_limit,
                                                input real  voltage_set,
                                                input real  initial_soc,
                                                input real  update_time);
```

As it is possible to note from the figure, the function is imported specifying the name of its formal arguments. The formal arguments obviously reflects the same variables described in the previous section in which the function `update_battery_model()` has been presented. Let's identify these arguments. The arguments `battery_current` and `battery_voltage` represent the charging current and the output voltage of the battery. As described in the previous section, these signals are passed to the function as pointers, in order that the function could modify them also in the SistemVerilog world. They represent an output for the battery module, that rather is driven by the signals `current_limit` and `voltage_set`, declared as input. These signals specify respectively the current and the voltage limits with which the function is called. The signal `initial_soc` represents the initial SOC with which the battery model is initialized at the first call to the function. Finally, `update_time` is the signal that represents the interval passed between to consecutive calls to the function. It is basically the interval in which the battery current is integrated to compute the new model state between to consecutive steps of state update.

All these formal arguments of the function have to be specified when the function is called. So, obviously, the arguments `battery_current` and `battery_voltage` of the function are mapped respectively to the nodes `i_batt` and `v_batt` of the SystemVerilog module. Similarly, `current_limit` and `voltage_set` are mapped respectively to the nodes `i_limit` and `v_set` of the SystemVerilog module. Let's consider `initial_soc` and `update_time`. They are mapped to `state_of_charge` and `t_update`. The values of this nodes are specified within the SystemVerilog module inside an `initial` block, in order to be initialized at the beginning of the simulations.

```
initial
    begin
        state_of_charge = INITIAL_SOC;
        t_update        = MODEL_UPDATE_TIME;
    end
```

The initialization values, respectively `INITIAL_SOC` and `MODEL_UPDATE_TIME`, are declared within the SystemVerilog module as parameters. In this mode, the initial value of the SOC and the value of the update time can be specified when the SystemVerilog battery module is instantiated in an higher level module or in a test bench.

```
// Battery module parameters.
parameter INITIAL_SOC       = 0.0;
parameter MODEL_UPDATE_TIME = 0.1;
```

Let's describe the mechanism with which the function `update_battery_model()` is called inside the SystemVerilog module. Once that the function has been imported in the SystemVerilog world using the appropriate construct, it can be called as a normal C function. So, an `always` block can be written to take care of calling the function at every rising edge of the input clock `sim_clock`, defined between the inputs of the SystemVerilog module. The `always` block is reported below.

```
// Call update_battery_model() every sim_clock positive edge.
always@(posedge sim_clock)
        state_of_charge     = update_battery_model(i_batt,
                                                   v_batt,
                                                   i_limit,
                                                   v_set,
                                                   state_of_charge,
                                                   t_update);
```

As shown, the function returns the current SOC. The return value is assigned at each rising edge of the `sim_clock` to the output node `state_of_charge`.

The SystemVerilog battery module can be easily represented as a block diagram. The diagram is shown in figure 55.

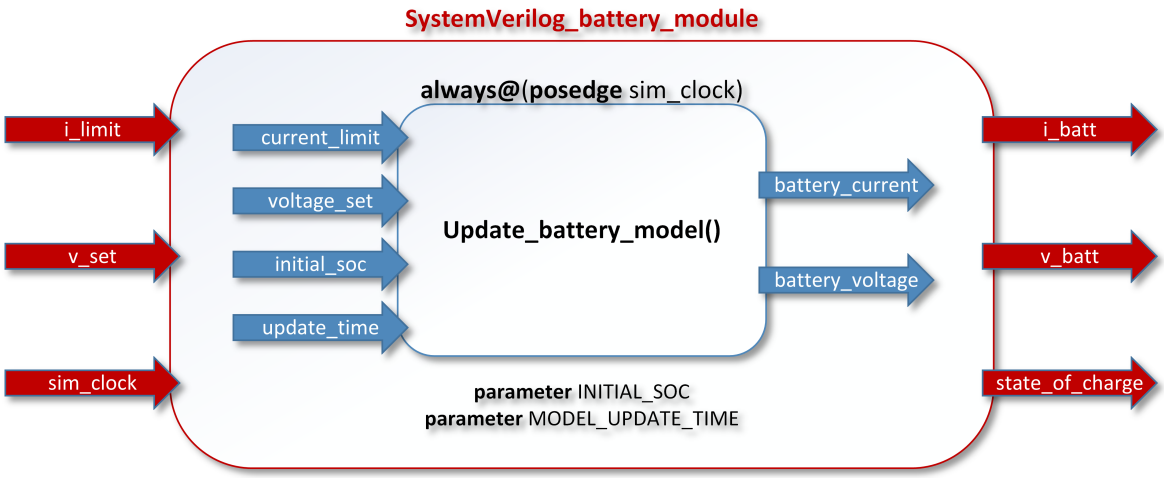

Figure 55: SystemVerilog battery module diagram.

5.3.4 SystemVerilog battery module simulation

Let's move to the SystemVerilog battery module simulation. As done for the other model, the first simulation is performed using as model input the current extracted from the pulsed current test. To run the simulation, a SystemVerilog test bench has been set up. The test bench contains only two modules. The first is an instance of the battery module while the second is a clock generator that generates the signal `sim_clock`. Below, the list of the signals used in the test bench is reported.

```
// Battery model parameters.
real i_battery;
real v_battery;
real state_of_charge;
real i_limit;
real v_set;

// Clock for model simulation.
logic sim_clock;

// File handler to import the measured current.
integer data_file;

// File handler to save simulation results.
integer output_file;
```

The signals `i_battery`, `v_battery` and `state_of_charge` represents again, at the top level, the battery module current, voltage and state of charge. Once again, also `i_limit` and `v_set` represents the current and the voltage to set with which the battery module is driven. The signal `sim_clock` represents the clock that drives the battery module. `data_file` and `output_file` are file identifiers. The first is used to represent the file containing the input current relative to the pulsed current test while the second is used for writing the simulation results.

Let's consider the battery module instance.

```
// sv_battery_module instance.
sv_battery_module #(.INITIAL_SOC (100.0), .MODEL_UPDATE_TIME (0.1))
    batt_module(.i_batt             (i_battery),        // Output.
                .v_batt             (v_battery),        // Output.
                .state_of_charge    (state_of_charge),  // Output.
                .i_limit            (i_limit),          // Input.
                .v_set              (v_set),            // Input.
                .sim_clock          (sim_clock));       // Input.
```

It is instantiated specifying for the parameter `INITIAL_SOC` the value 100 % and for the parameter `MODEL_UPDATE_TIME` the value 0.1 s. These parameters reflects the battery parameters used during the pulsed current test. In fact, the initial condition in which the battery lies at the beginning of the PCT is with the full SOC. Moreover, the sampling time used for the data acquisition during the realization in laboratory of the test is just 0.1 s. After specifying the values of the module parameters, in the instance the connections between the signals of the test bench and those relative to the module need to be defined.

Let's consider the `sim_clock` signal generation.

```
// Generate simulation clock.
always
    #5 sim_clock    = ~ sim_clock;
```

The signal `sim_clock` is declared as logic. Every five cycles of the `timeunit` the signal changes its value from 0 to 1 or vice versa. As `timeunit`, the value of 10 µs has been chosen.

```
timeunit 10us;
```

Consequently, the period of the signal `sim_clock` in the simulation is of 0.1 ms. Let's introduce some definitions:

- T_{update} represents the real time that passes between two consecutive battery state updates.
- T_{sim_clock} represents the period of the signal `sim_clock`.
- T_{real} is the real time passed during the execution of the pulsed current test.
- $T_{simulation}$ is the time used in simulation to realize the pulsed current test.
- $CallsNumber$ is the number of calls of the function `update_battery_model()`.

It is possible to write:

$$CallsNumber = \frac{T_{real}}{T_{update}} = \frac{T_{simulation}}{T_{sim\_clock}} \tag{46}$$

As introduced above, in this case $T_{update} = 0.1\,\text{s}$ and $T_{sim\_clock} = 0.1\,\text{ms}$. Consequently:

$$T_{simulation} = \frac{T_{real}}{1000} \tag{47}$$

So, maintaining the number of the calls to the function `update_battery_model()`, it is possible to shorten the simulation time in respect to the real experimental time. This has been realized in this test bench. The real duration of the pulsed current test is of about 45 h. Considering $T_{update}/T_{sim\_clock} = 1000$, the duration of pulsed current test in simulation is of about 160 s.

This behaviour of the battery module can be exploited for shortening the simulation time can be used not only for the pulsed current test, but for all the test that you want.

There is another aspect to explain. Maintaining constant the test real duration, it is possible to reduce the number of the calls to the function `update_battery_model()` increasing the value of T_{update}. In this case, differently from the previous, during simulation, the error introduced by the discretization of the battery model grows.

Let's return to the description of the SystemVerilog test bench. The description of the process that manage the test bench is realized again using an `always` block. The block is reported below.

```
// Call update_battery_model() every sim_clock positive edge.
always@(posedge sim_clock)
    begin
        $fscanf(data_file, "%.6f\n", i_limit);
        if(!$feof(data_file))
            begin
                if(i_limit > 0)
                    v_set = 10.0;
                else if(i_limit < 0)
                    v_set = 0.0;
                else
                    v_set = v_set;
                $display("v_battery = %.6f\n", v_battery);
                $display("i_battery = %.6f\n", i_battery);
                $display("soc_battery = %.6f\n", state_of_charge);
                $display("--------------------------------\n");
                $fwrite(output_file, "%.6f\t", state_of_charge);
                $fwrite(output_file, "%.6f\t", i_battery);
                $fwrite(output_file, "%.6f\n", v_battery);
            end
    end
```

At every rising edge of the signal `sim_clock`, the value of the node `i_limit` is set to the corresponding value of the current during the pulsed current test. As mentioned above, the samples of the current are read from the variable `data_file`, that is an identifier that represents the file that contains the profile of the measured current. Moreover, again at every rising edge of `sim_clock`, the function `update_battery_model()` is called inside the SystemVerilog module of the battery. Then the function updates the battery model state and the variables `i_batt`, `v_batt` and `state_of_charge`, that correspond in the test bench respectively to `i_battery`, `v_battery` and `state_of_charge`. Finally the values of these variables can be read, printed on screen and saved on file. To save the samples, the file identifier `output_file` is used.

Once the test bench has been presented, the simulation can be launched. The simulation is done on Cadence Incisive Enterprise Simulator. The results are reported in the following figure 56.

Figure 56: Digital/mixed-signal battery model simulation on PCT.

The output file produced during the simulation can now be imported in Matlab in order to evaluate the error $V_{measure} - V_{simulation}$, as described for the other battery models.

The figures 57 and 58 show the obtained results.

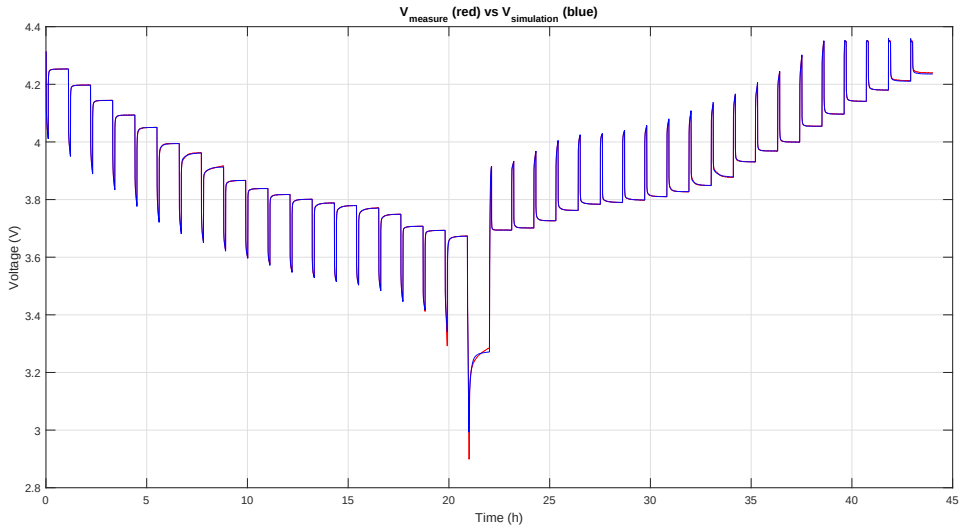

Figure 57: Digital/mixed-signal model results on PCT exported in Matlab.

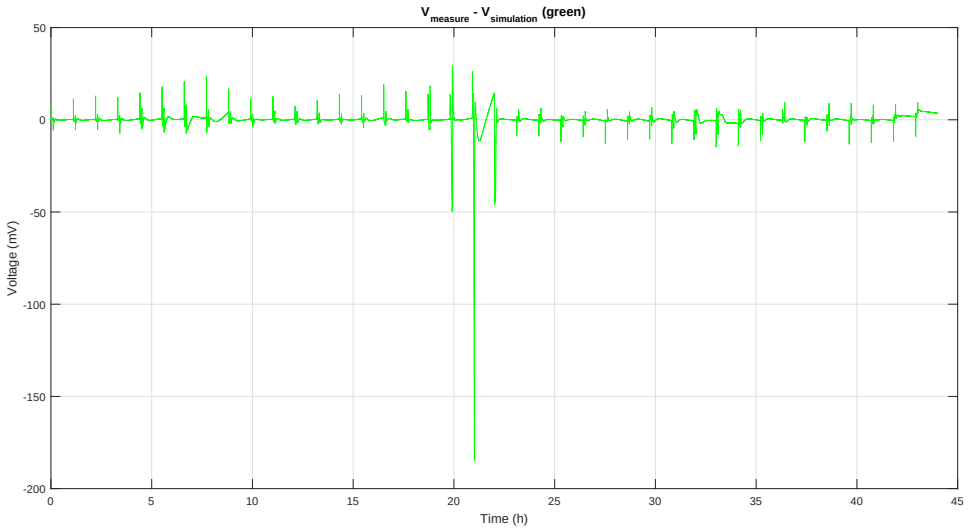

Figure 58: Digital/mixed-signal model error on PCT computed in Matlab.

Finally, again using Matlab, the numerical value of the error $V_{measure} - V_{simulation}$ can be extracted. The error is reported in table 6.

	Maximum error	Maximum percentage error	Root mean square error	Root mean square percentage error
Simulink model	183.5 mV	6.33 %	4.7 mV	0.15 %
Analog model	193.7 mV	6.68 %	5.7 mV	0.18 %
Digital model	184.8 mV	6.37 %	4.4 mV	0.14 %

Table 6: Comparison of the simulation results of the models on PCT.

Once again, it is possible to see that the error introduced by the digital model of the battery is comparable with those of the Simulink model and of the analog model and so, also this model can be considered correctly realized.

6 Model validation

6.1 Validation description

Once the battery models have been implemented in the different environments and tested on the pulsed current test, it is necessary to verify their behaviour when they are used to simulate a different test than the PCT.

The validation aims at verifying that the behaviour of the implemented models is similar to that of the real battery, also in a different test in respect to that with whom the battery model parameters have been extracted.

The validation procedure is based on the test described in [20]. The validation test has been chosen in order to represent a possible profile of the battery current during the battery normal operation.

For this reason, the test consists of a series of current pulses, both of discharge and charge, of various duration and amplitude followed by resting period.

The profile of the current is reported in figure 59.

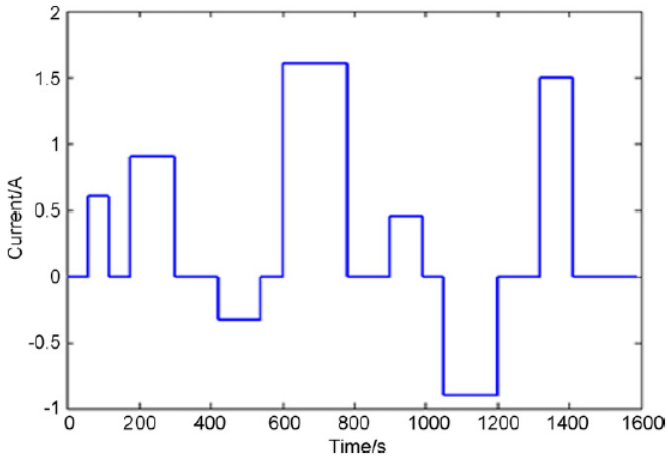

Figure 59: Validation test procedure [20].

Let's consider the figure. If the current profile was realized as it is described in the paper cited above, then the variation of the state of charge of the battery during the test would be less than 5 %. A variation of the SOC of 5 % is not sufficient for the realization of an acceptable validation test.

For this reason, the duration of the pulses and that of the pauses of the test are quadruplicated and the whole profile is repeated three times to realize an acceptable validation test. Doing this, the difference between the battery SOC at the beginning of the test and that at the test end is of about 50 %.

Once the validation test has been chosen, it has to be firstly realized in laboratory using the real battery. The experimental setup used for the validation test is the same

used to realize the initial battery training and the pulsed current test. For the detailed description of the experimental setup, refer to section 3.

The test is composed by two different phases:

- Firstly a complete discharging cycle is performed up to a voltage of 2.90 V using a current equal to C/2, followed by an hour of pause in which the current is set to zero. After, the battery is subjected to a complete charging cycle, up to a voltage of 4.35 V, again using a current of C/2, followed by another pause of an hour. This initial phase allows the test to start with the battery in a well known condition of SOC equal to 100 %.
- Then the real validation test is performed. In this phase the battery is driven with the current profile described above.

The results of the validation test are reported in the figures 60, 61 and 62. Firstly, in figure 60, the voltage at the battery terminals is shown.

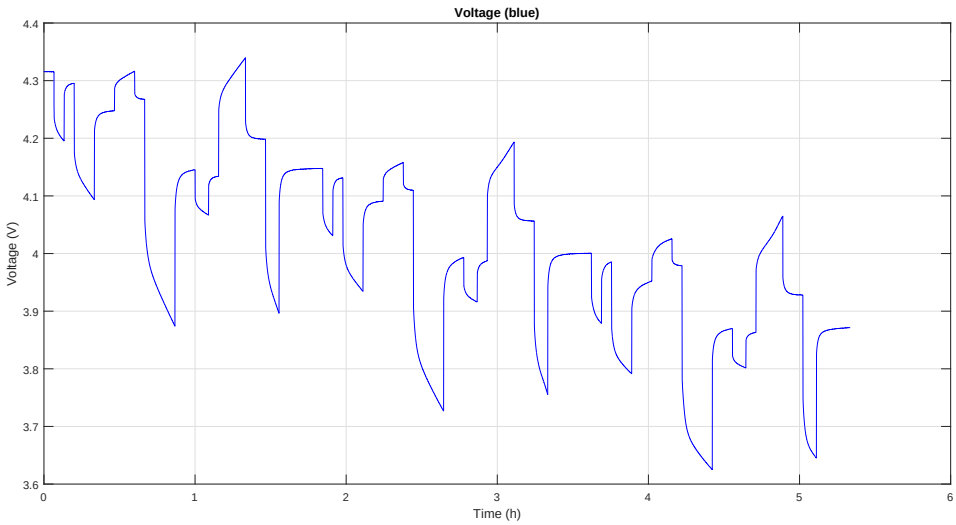

Figure 60: Validation test voltage.

Figure 61 shows instead the trend of the battery current. Let's observe that the current is changed in the sign in respect to that presented in figure 59. In this case, the current is considered positive in charge, while in the paper, the discharging current is positive.

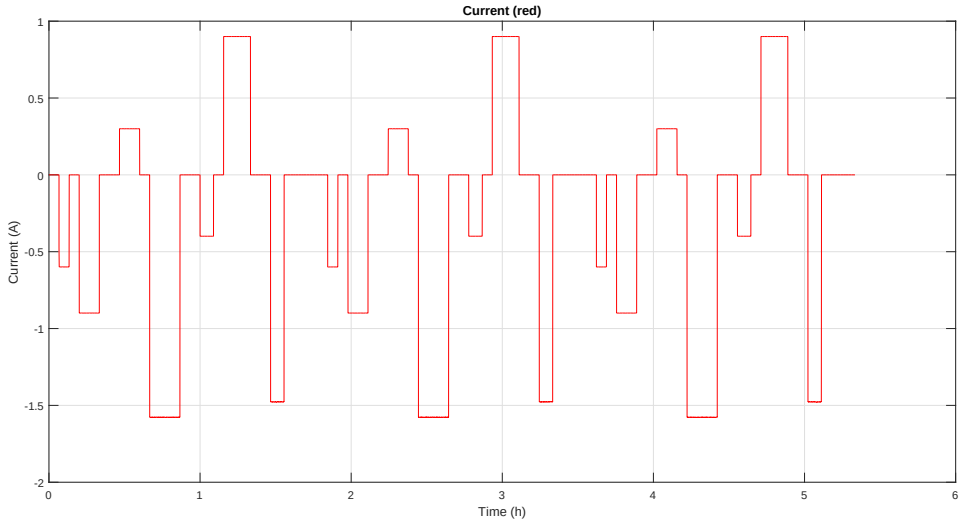

Figure 61: Validation test current.

Finally, figure 62 shows the trend of the state of charge during the validation test. Let's observe that the battery is fully charged at the beginning of the test.

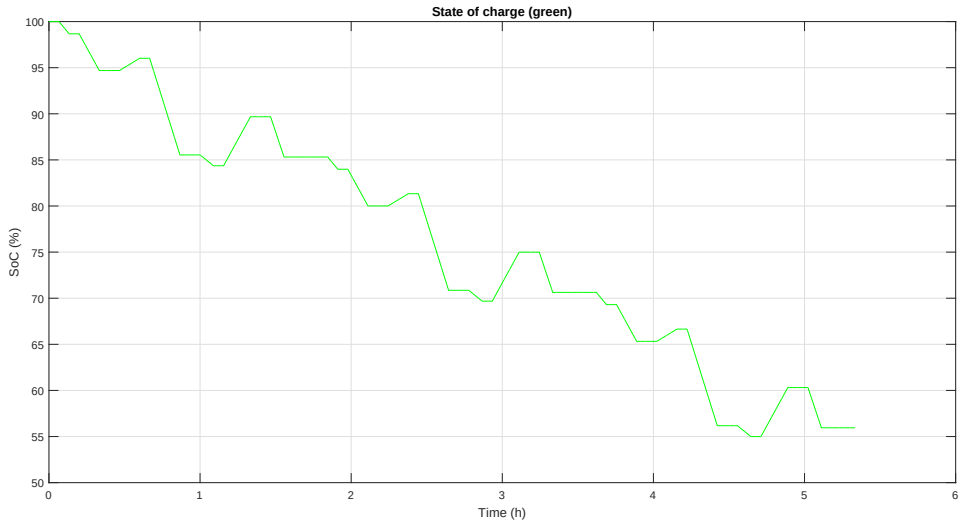

Figure 62: Validation test state of charge.

Once the test has been realized on the real battery and the results have been collected, the current measured during the validation test can be extracted and then used as input to drive battery models implemented in the different environments. So, firstly the Simulink model, followed by the analog and then by the digital are simulated in order again to test their behaviour.

After the model simulation, the output voltage can extracted in order to be compared with the voltage acquired during the validation test in laboratory. Finally, for each battery model, the error between the measured and the simulated voltage can be evaluated.

In the following section, the validation procedure is presented for each of the implemented battery models.

6.2 Simulation on the validation test

6.2.1 Simulink model validation

The first simulation is performed on the Simulink model. The results of the simulation are shown in figures 63 and 64.

Figure 63 shows the trend of the simulated voltage, in blue, in respect to the measured, in red.

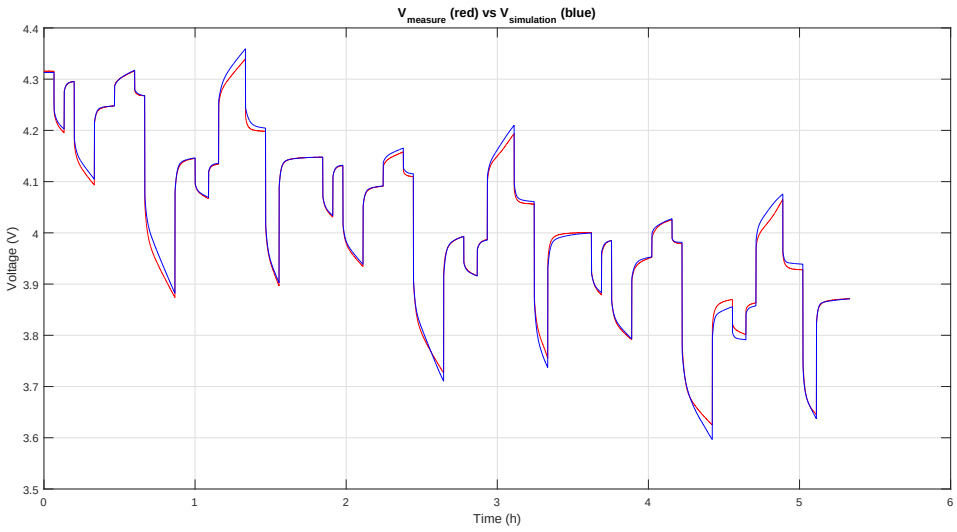

Figure 63: Simulink model validation.

Figure 64, instead, shows the trend of the error $V_{measure} - V_{simulation}$.

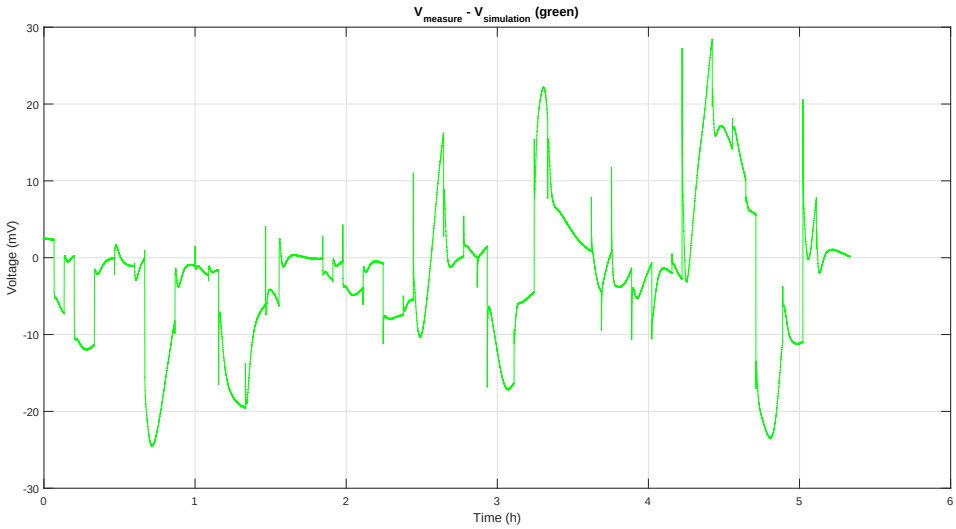

Figure 64: Simulink model error on validation test.

One time again, the error $V_{measure} - V_{simulation}$ can be evaluated. The error is reported in the table 7 below.

	Maximum error	Maximum percentage error	Root mean square error	Root mean square percentage error
Simulink model	28.5 mV	0.62 %	9.3 mV	0.24 %

Table 7: Error introduced in the validation test by the Simulink model.

6.2.2 Analog model validation

As seen for the Simulink model, also the analog model is simulated on the validation test. The simulation is performed in Cadence Spectre Circuit Simulator. For the complete description of how the simulation is performed in the Cadence Virtuoso environment, refer to the section 5.

The simulation results are reported in figure 65.

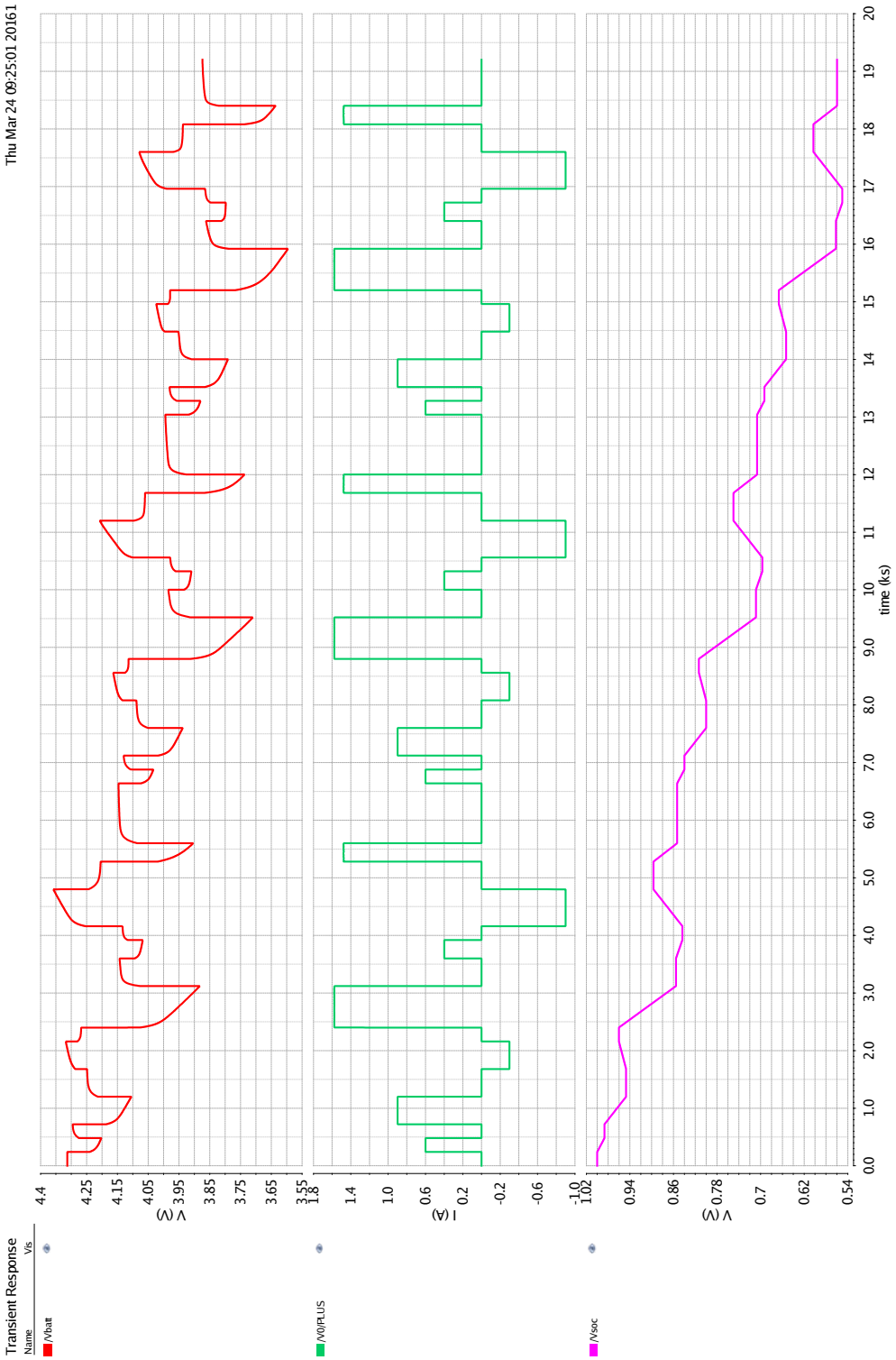

Figure 65: Analog model validation.

The simulation results of the analog model can be exported from Cadence and then imported in Matlab in order to be compared with the results obtained with the Simulink model and in order to extract the error between the analog model and the real battery.

The results are shown in the figures 66 and 67. In particular, figure 66 shows the comparison between the profiles of the measured and the simulated voltage, while figure 67 shows the error $V_{measure} - V_{simulation}$.

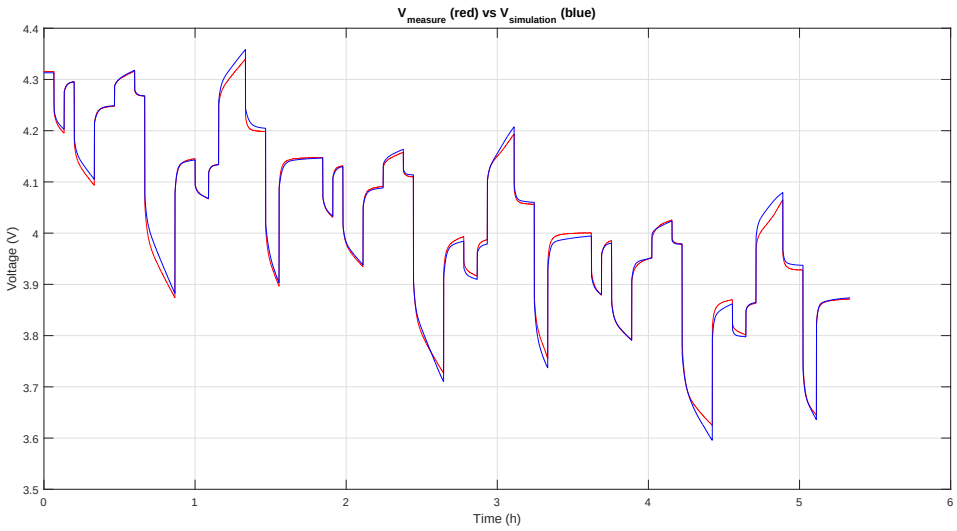

Figure 66: Analog model validation test exported in Matlab.

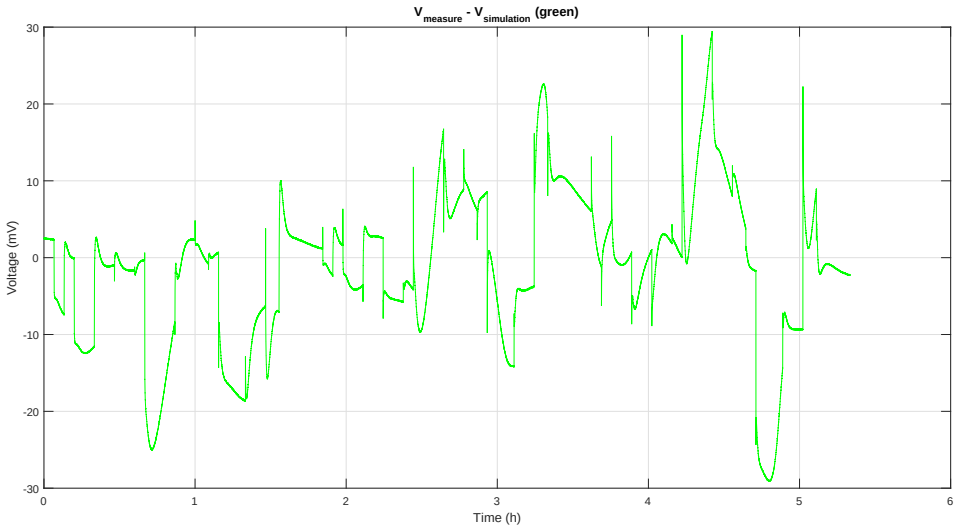

Figure 67: Analog model validation test error computed in Matlab.

Again using Matlab, it is possible to extract the error introduced by the analog model when it is driven by the input current extracted from the validation test.

The results appear in the table 8 below, compared with that relative to the Simulink model.

	Maximum error	Maximum percentage error	Root mean square error	Root mean square percentage error
Simulink model	28.5 mV	0.62 %	9.3 mV	0.24 %
Analog model	29.5 mV	0.73 %	9.8 mV	0.25 %

Table 8: Simulink model vs analog model on validation test.

Let's observe that the errors introduced by the Simulink and by the analog model are comparable.

6.2.3 Digital/mixed-signal model validation

Finally, the digital/mixed-signal model is simulated. As it has been for the simulation of the pulsed current test, also in this case the simulation is performed in Cadence Incisive Enterprise Simulator. The test bench used to perform the validation of the model is basically the same seen in the previous section 5 in which the digital battery model has been simulated on the pulsed current test.

The simulation results are reported in the figure 68.

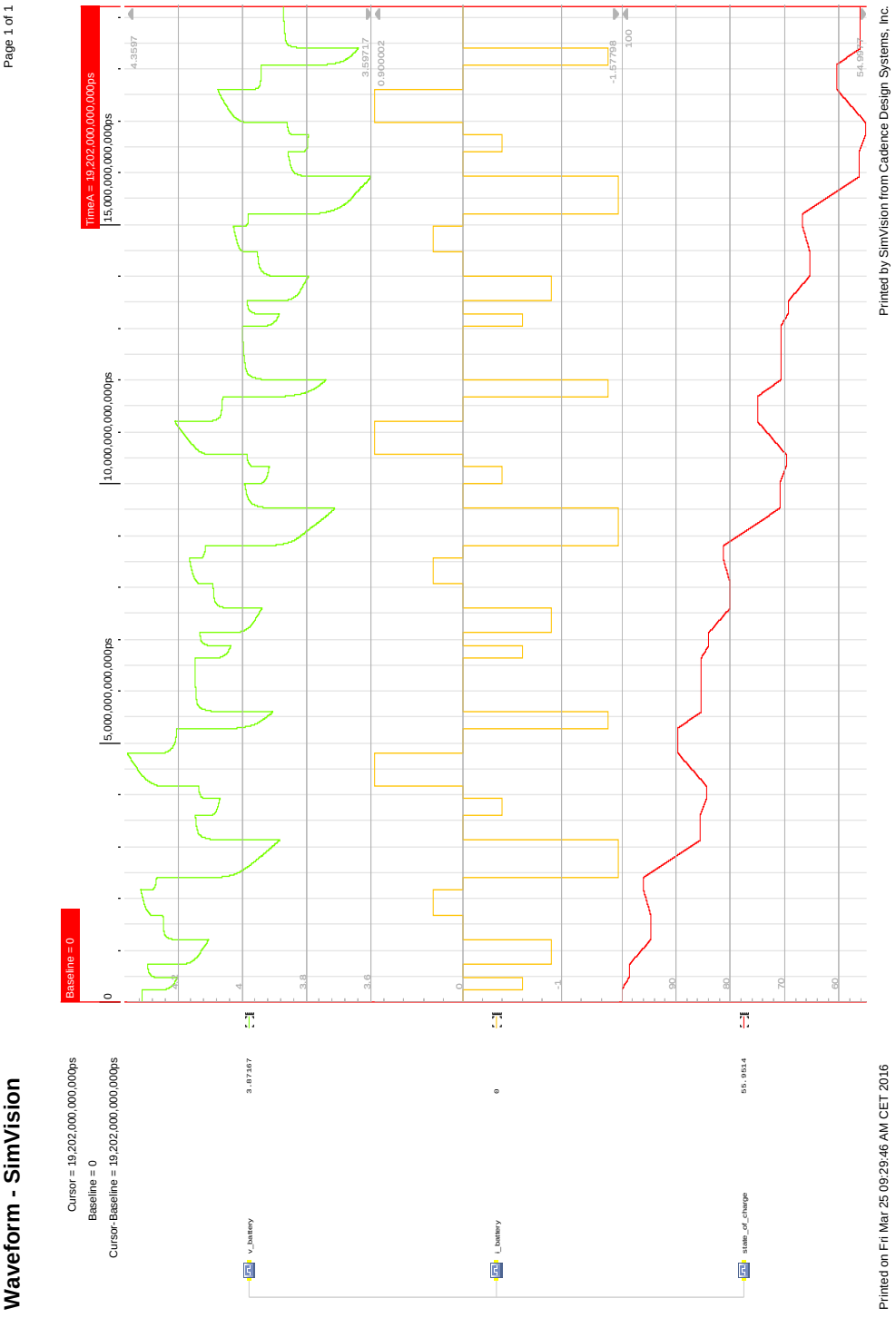

Figure 68: Digital/mixed-signal battery model validation.

The simulation results can now be exported from Cadence and imported in Matlab in order to be compared with those of the other models and in order to extract the error introduced by the digital/mixed-signal model.

Figures 69 and 70 show the result obtained simulating the digital model on the validation test.

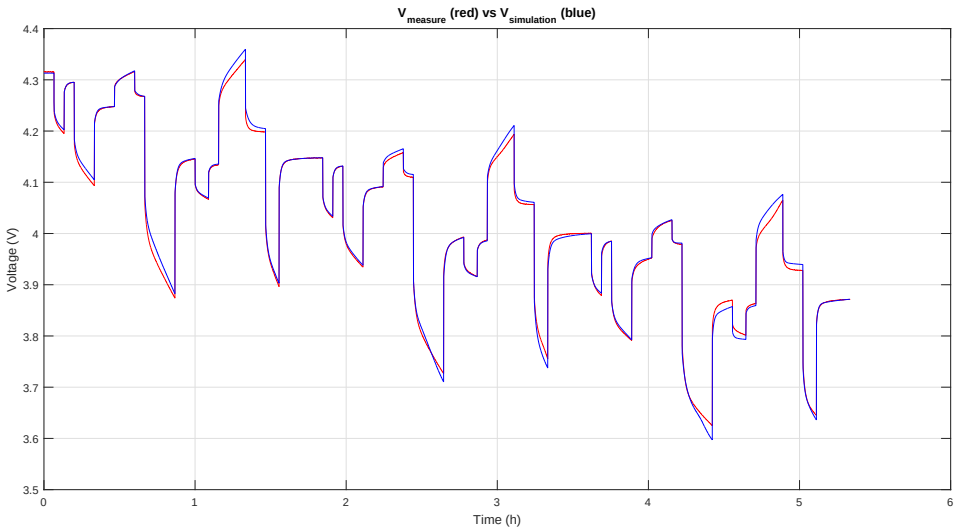

Figure 69: Digital/mixed-signal model validation results exported in Matlab.

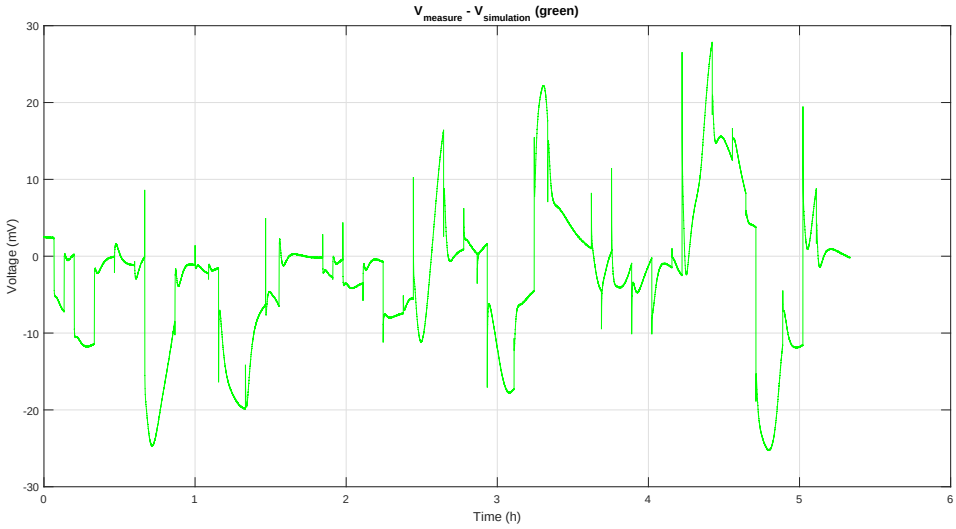

Figure 70: Digital/mixed-signal model validation error computed in Matlab.

Finally, also for this model, the error is extracted and compared with the error introduced by the two other models.

Table 9 summarizes the behaviour of the Simulink model, of the analog model and of the digital model when they simulate the validation test.

	Maximum error	Maximum percentage error	Root mean square error	Root mean square percentage error
Simulink model	28.5 mV	0.62 %	9.3 mV	0.24 %
Analog model	29.5 mV	0.73 %	9.8 mV	0.25 %
Digital model	27.9 mV	0.63 %	9.4 mV	0.24 %

Table 9: Comparison between the validation errors of the three battery models.

As introduced in section 1, the root mean square error and the maximum error between the measured and the simulated voltage, evaluated on the validation test, present a value less respectively than 0.3 % and than 0.8 %.

This attests that the implemented models are able to predict the battery behaviour accurately, in all the three different simulation environments.

In the following section 7, the conclusions of the work are presented.

7 Conclusions

In this work, a complete modelling process has been presented for a portable electronic device Li-ion battery. The purpose of the thesis, as presented at the beginning of the document, was the development of a behavioural battery model, aimed at being used in system-level simulations. The battery model, in fact, will serve in turn for the design and the optimization of battery management systems, battery chargers, and for the development of fuel gauge algorithms. To account for the multi-domain aspects of the design of these systems, three different versions of the battery model have been realized in three different simulation environments. So, firstly, a high-level model has been implemented in the Matlab/Simulink environment, the, an analog model has been realized in Cadence Virtuoso, and and finally a digital/mixed-signal model has been implemented using SystemVerilog in the Cadence Incisive environment.

For the modelling process, a well known two time-constant model has been used to represent the behaviour of a lithium-ion battery.

Let's summarize the workflow that has allow to develop these battery models. Firstly, the cell has been characterized using a well defined experiment, the pulsed current test. To this end, a classical experimental setup, which is based on a source-meter unit, able to provide the characteristics both of a power supply and of an electronic load, has been used.

Then, the parameters related to the battery equivalent circuit model have been extracted, firstly using the conventional estimation method, and then by introducing an optimized technique.

Once the ECM parameters have been estimated, they have been used for the implementation of the battery models respectively in Matlab/Simulink, in Cadence Virtuoso, and Cadence Incisive.

Then, after the implementation of the different models, an initial verification of their behaviour has been performed on the pulsed current test, used for the battery characterization aimed at the parameter extraction.

The results obtained in the first simulation of the battery models on the pulsed current test have been good. The root mean square error between the voltage measured during the test on the real battery and the simulated voltage, in fact, has been lower than 0.2 % for all the three battery models.

Finally, a real validation process has been performed, in which the behaviour of the battery models has been verified on a different test in respect to the pulsed current test. Also in this case, the models have been able to predict the real battery behaviour with good accuracy, testified by the close agreement between measures and simulations. The value of root mean square error, in this case, has been lower than 0.3 % for all the three battery models, a very good value.

Let's now consider the possible future developments. The method for the improvement of the battery models is based on a more detailed and extensive characterization of the battery to be model.

Considering again the pulsed current test as characterization test, the battery could

be tested at different temperatures controlled using a thermal chamber. Moreover, the battery characterization could be repeated using also different current values.

In this way, the battery ECM would present, in addition to the already presented dependency from the state of charge and from the current direction, the dependency from the temperature and that from the current amplitude [21], [22].

In this case, the parameter estimation algorithm does not change, but it should be repeated for each combination of the temperature and the current values used in the battery PCT, making however the parameter estimation more complex.

A more marked difference may regard instead the implementation of the battery models. The parameter lookup tables of the battery models, in fact, become tri-dimensional.

This would make obviously the battery models more accurate and so their behaviour more similar to that of the real cell, but, on the other hand, in this way the battery models would be also dramatically more complex, because of the tri-dimensional interpolation of the parameter lookup tables.

Consequently, also the battery model simulation times would be greater and this could complicate the design of the devices, like battery management systems and battery chargers, for whom the battery models have been realized.

Therefore, it is a matter of finding, from time to time, the right compromise between the accuracy of the battery model and its complexity.

Acronyms

EOD	End-of-discharge
EOC	End-of-charge
SOC	State of charge
OCV	Open-circuit voltage
LCO	Lithium cobalt oxide
LMO	Lithium manganese oxide
NMC	Lithium nickel manganese cobalt oxide
LFP	Lithium iron phosphate
NCA	Lithium nickel cobalt aluminum oxide
LTO	Lithium titanate
NiCd	Nickel–cadmium
NiMH	Nickel–metal hydride
ECM	Equivalent circuit models
CC	Constant-current
CV	Constant-voltage
PCT	Pulsed current test

References

[1] D. Linden and T. B. Reddy, "*Handbook of batteries (3rd Edition)*", McGraw-Hill, 2002.

[2] V. Pop, H. J. Bergveld, D. Danilov, P. P. L. Regtien, P. H. L. Notten, "*Battery management systems: accurate state-of-charge indication for battery-powered applications*", Springer, 2008.

[3] batteryuniversity.com, "*BU-205: Types of Lithium-ion*".

[4] L. Gao, S. Liu, R. A. Dougal, "*Dynamic lithium-ion battery model for system simulation*", in IEEE Transactions on Components and Packaging Technologies, vol. 25, no. 3, September 2002.

[5] V. Ramadesigan, P. W. C. Northrop, S. De, S. Santhanagopalan, R. D. Braatz, and V. R. Subramanian, "*Modeling and simulation of lithium-ion batteries from a systems engineering perspective*", in Journal of The Electrochemical Society, 159 (3) R31-R45 (2012).

[6] J. Newman, K. E. Thomas, H. Hafezi, D. R. Wheeler, "*Modeling of lithium-ion batteries*", in Journal of Power Sources 119–121, 2003, pp. 838–843.

[7] M. Chen and G. Rincon-Mora, "*Accurate electrical battery model capable of predicting runtime and i-v performance*", in IEEE Transactions on Energy Conversion, Jun. 2006, pp. 504–511.

[8] D. Andre, M. Meiler, K. Steiner, H. Walz, T. Soczka-Guth, D. U. Sauer, "*Characterization of high-power lithium-ion batteries by electrochemical impedance spectroscopy. II: Modelling*", in Journal of Power Sources 196, 2011, pp. 5349–5356.

[9] mathworks.com/products/matlab.

[10] mathworks.com/products/simulink.

[11] cadence.com/products/cic.

[12] doulos.com/knowhow/sysverilog, "*What is SystemVerilog?*".

[13] cadence.com/products/fv/enterprise_simulator.

[14] python.org/about.

[15] S. Abu-Sharkh and D. Doerffel, "*Rapid test and non-linear model characterisation of solid-state lithium-ion batteries*", in Journal of Power Sources, vol. 130, pp. 266–274, May 2004.

[16] L. W. Yao, J. A. Aziz, P. Y. Kong, N. R. N. Idris, "*Modeling of lithium-ion battery using MATLAB/simulink*", in Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE, pp. 1729-1734.

[17] R. Jackey, M. Saginaw, P. Sanghvi, J. Gazzarri, T. Huria and M. Ceraolo, "*Battery model parameter estimation using a layered technique: an example using a lithium iron phosphate cell*", in SAE Technical Paper 2013-01-1547, 2013, doi:10.4271/2013-01-1547.

[18] F. Pêcheux, C. Lallement, A. Vachoux, "*VHDL-AMS and Verilog-AMS as alternative hardware description languages for efficient modeling of multidiscipline systems*", in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 24, no. 2, February 2005.

[19] C. Spear, G. Tumbush, "*SystemVerilog for verification (3rd Edition)*", Springer, 2012.

[20] X. Hu, S. Li, H. Peng, "*A comparative study of equivalent circuit models for li-ion batteries*", in Journal of Power Sources 198, 2012, pp. 359–367.

[21] F. Baronti, G. Fantechi, E. Leonardi, R. Roncella, and R. Saletti, "*Enhanced model for lithium-polymer cells including temperature effects*", in IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society. IEEE, Nov. 2010, pp. 2329–2333.

[22] F. Baronti, G. Fantechi, E. Leonardi, R. Roncella, and R. Saletti, "*Effective modeling of temperature effects on lithium polymer cells*", in 2010 17th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Dec. 2010, pp. 990–993.